UAH

Definición de arquitecturas eficientes basadas en FPGAs para el procesamiento de señales biomédicas.

Máster Universitario en Sistemas Electrónicos Avanzados.

Sistemas Inteligentes

Departamento de Electrónica

Presentado por:

D. JAVIER ROBLES MACHÍN

Dirigido por:

Dr. ÁLVARO HERNÁNDEZ ALONSO

Dª. RAQUEL GUTIÉRREZ RIVAS

Alcalá de Henares, a 20 de Septiembre de 2015

Contenido

Abstract

The development of algorithms for biomedical signal processing has been the cause of research for some time, but it was not until the early 90s that the technology is put on a par with these investigations by providing processors with enough computing capacity in real time to carry them out.

This paper will seek to provide a solution, especially directed to basic computers without a significant amount of resources for detecting peaks on an electrocardiogram (ECG). For this an intensive study of the biomedical signal for the further development and implementation of an algorithm in a programmable logic device (FPGA) will be performed.

Resumen

El desarrollo de algoritmos para el procesamiento de señales biomédicas ha sido causa de investigación desde hace tiempo, sin embargo no fue hasta principio de los 90 que la tecnología se puso a la par con estas investigaciones aportando procesadores con la suficiente capacidad de cálculo en tiempo real como para llevarlos a cabo.

En este trabajo se pretenderá aportar una solución, sobre todo dirigida a sistemas básicos que no tengan una cantidad de recursos importante, para la detección de picos en una señal electroicardiográfica (ECG). Para ello se realizará un estudio intensivo de esta señal biomédica para el posterior desarrollo e implementación de un algoritmo en un dispositivo de lógica programable (FPGA).

Lista de Figuras

Cuando la memoria incluya referencias constantes a diversos acrónimos (más de una decena), puede ser recomendable utilizar un índice de acrónimos que recuerde su significado. En este índice los acrónimos se ordenarán alfabéticamente, adjuntando su definición exacta (en inglés o español, en función de su origen) en la tabla adjunta. Ejemplos:

Lista de tablas

Capítulo 1. Introducción

Se puede definir el término señal como el medio a través del cual se transmite la información. La adquisición y tratamiento de esta información permiten aprender y conocer diversos aspectos de la fuente a partir de la cual es generada.

Extrapolando esto al campo médico, se encuentran las lamadas bioseñales. Cada tipo de célula presenta un comportamiento eléctrico particular y este comportamiento a su vez revela información sobre el órgano en el que se encuentra. La captación de estas bioseñales y su posterior interpretación permiten descubrir diversos problemas o patologías en un sujeto, revelando de esta forma su estado de salud.

Estas señales eléctricas generadas por el organismo suelen tener una serie de características comunes independientemente de su fuente de origen. Entre las más importantes se encuentran por ejemplo que son de muy pequeña magnitud (del orden de los µV hasta algunos pocos mV) y de una relación señal-ruido alta, debido a esto, y con el objetivo de poder extraer la información buscada sin que ésta se corrompa en el proceso, la adquisición de este tipo de señales va acompañada de un procesado en tiempo real que nos permita obtener la información de la mejor forma posible.

Este trabajo se centrará en el estudio e implementación sobre FPGA de un algoritmo para el procesado de una de las bioseñales más importantes, éstas son aquellas denominadas como ECG o electro-cardiográficas, las cuales no son más que la representación gráfica de la actividad eléctrica del corazón.

Los electrocardiogramas son el instrumento principal de la electrofisiología cardiaca y tienen una función de gran relevancia en el mundo médico para llevar a cabo el cribado y diagnóstico de diversas enfermedades cardiovasculares, alteraciones metabólicas y la predisposición a una muerte súbita cardiaca. Actualmente, también tiene otros usos aparte del meramente médico ya que son de gran utilidad para conocer la duración del ciclo cardiaco. Esto puede ser, por poner un ejemplo, un gran aliado en el entrenamiento de deportistas que realicen pruebas aeróbicas, como por ejemplo el ciclismo o atletismo.

El procedimiento a seguir generalmente en la adquisición de estas señales se puede resumir en los siguientes pasos:

- Detección de la bioseñal a través de sensores colocados lo más cerca posible de la fuente. El sensor convertirá la magnitud física en una señal eléctrica actuando así de puente entre el sistema biológico y el instrumento que recogerá la información.
- Amplificación y filtrado de la bioseñal. Como ya se ha comentado estas señales se caracterizan con tener una elevada SNR. Debido a esto es necesario una etapa de acondicionamiento.
- La señal es discretizada mediante un ADC para su posterior procesado.

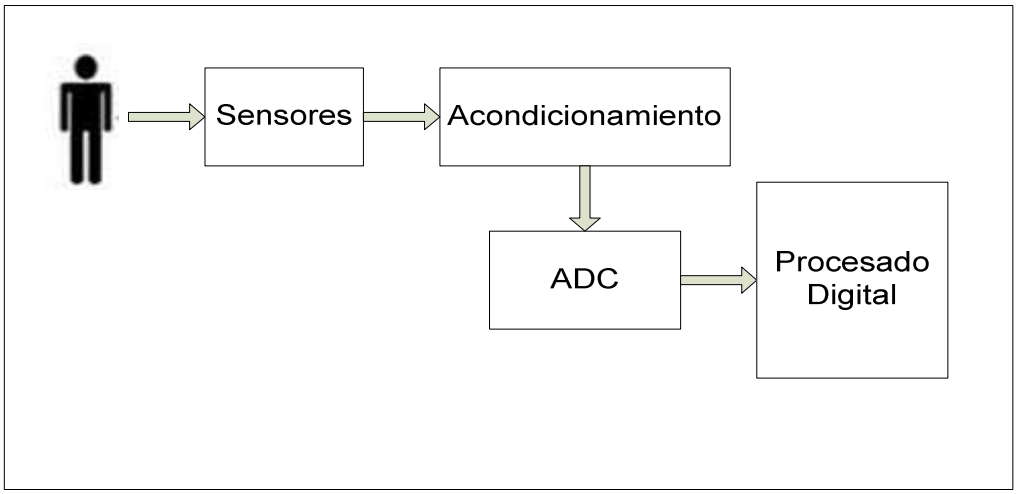

Figura 1.1. Esquema genérico del proceso de captación de una señal biológica.

Es importante remarcar que la conversión analógico digital de la señal presenta varias ventajas para las aplicaciones biomédicas. Algunas de estas ventajas son por ejemplo: el almacenamiento de información, lo que permitiría llevar un historial clínico de un paciente en un dispositivo de pequeñas dimensiones (tarjeta SD, pendrive, CD...); la transmisión de la información entre dispositivos, ya se encuentren próximos o lejanos, o, por ejemplo, el procesado en tiempo real de la señal en cuestión, que permite cribar de forma rápida y sencilla la información concreta que se desea de ella.

Sin embargo a la hora de realizar un sistema como el que se está llevando a cabo, o (en general) a la hora de trabajar con cualquier señal biomédica, es necesario en primer lugar realizar un estudio detallado de las características de la señal que se

desea adquirir con el fin de caracterizarla lo mejor posible y realizar un sistema robusto, eficiente y preciso.

En este trabajo se intentará caracterizar lo mejor posible las señales electro-cardiográficas, por ser la bioseñal a procesar y se describirá la forma en la que es tratada para tal y como reza el título del trabajo, **definir una arquitectura eficiente basada en FPGA para el procesado de señales biomédicas**, más concretamente ECGs.

1.1. Contexto

El presente trabajo está enmarcado dentro de las investigaciones llevadas a cabo por el Departamento de Electrónica de la Universidad de Alcalá en el ámbito del procesado de señales biomédicas y más concretamente en el diseño de arquitecturas sobre dispositivos lógicos programables para el manejo y tratamiento de este tipo de señales.

El trabajo con bioseñales no es una tarea sencilla, ya que es necesario conocer los aspectos biológicos y eléctricos propios del organismo que las genera, en nuestro caso el ser humano.

Este trabajo pretende describir el conocimiento y los procedimientos necesarios para el diseño y la implementación de un sistema capaz de realizar de forma eficiente el procesado de una de estas señales, más concretamente, la señal electro-cardiográfica (ECG).

1.2. Estructura del documento

La estructura de este trabajo está marcada por las pautas dadas desde el Departamento de Electrónica de la Politécnica Superior de la UAH. A continuación, se describirán cuáles son los puntos más importantes que constituyen el trabajo y los aspectos más relevantes que se tratarán en cada uno de ellos.

1. **Capítulo Introducción**: En esta sección se realizará la introducción sobre el propio trabajo, comenzando por el contexto general y aproximándose de forma gradual a los aspectos más concretos que serán tratados. De esta forma se pondrán en antecedentes los dos pilares que sustentan esta investigación hablando del procesado de señales biológicas y de los

dispositivos lógicos programables para acabar describiendo las ondas ECG y uno de los algoritmos más populares que existen hoy en día para realizar el procesado de las mismas el algoritmo Pan&Tompkins [14].

Además de todo esto se describirán los objetivos buscados en este trabajo.

2. **Capítulos de desarrollo**: El desarrollo de este trabajo se llevará a cabo en dos partes. En la primera de ellas se realizará una descripción del algoritmo planteado, se realizarán los modelos en coma fija y coma flotante ayudándonos de la herramienta Matlab y se mostrarán las simulaciones de ambos modelos indicando las diferencias si las hubiese.

   En la segunda parte se indicará cómo se ha implementado el modelo antes descrito sobre un dispositivo FPGA, describiendo la arquitectura elegida, el diagrama de bloques, la elección del datapath y mostrando las diferentes simulaciones funcionales de cada uno de los bloques.

3. **Resultados**: En este apartado se expondrán los resultados obtenidos comparando las simulaciones temporales con la de los modelos realizados en Matlab.

4. **Conclusiones / trabajos futuros**: Se realizara una reflexión sobre trabajos pendientes y futuras vías de investigación.

Capítulo 2. Antecedentes

En este trabajo se manejarán dos temas que confluyen con el fin de dar solución al objetivo final del mismo: la definición de una arquitectura eficiente sobre FPGA para el procesado de señales biomédicas.

Por lo tanto es necesario, a la hora de hablar de antecedentes, hacer referencia a las dos vertientes:

- Por un lado, está la evolución del tratamiento de señales biológicas, haciendo especial hincapié en las señales electro-cardiográficas (que como ya se he apuntado con anterioridad, son en las que se centrará el estudio)
- Por el otro, es necesario conocer cómo han influido los dispositivos programables o FPGA en el desarrollo de sistemas de procesado y cómo será usado este background para la implementación del algoritmo que se desarrollará en el presente trabajo.

2.1. Tratamiento de señales biomédicas

Dentro del grupo de señales biomédicas se encuentran diversos tipos, como por ejemplo bioacústicas, biomagnéticas, de bioimpedancia o bioeléctricas. A lo largo de todo este trabajo el protagonismo lo tomará una señal perteneciente al último grupo de los citados y que se conoce como señal electro-cardiográfica. A lo largo de los siguientes puntos se intentará caracterizar de la forma más exacta posible, indicar qué características de la señal son importantes a la hora de realizar un tratamiento sobre ella y describir los procesos que la originan.

2.1.1. Antecedentes Históricos

La ingeniería ha aportado diversas soluciones tecnológicas en la adquisición de señales biomédicas desde que el doctor holandés Willem Einthoven [13] inventara allá por el 1901 el primer método práctico para adquirir las señales eléctricas producidas por el corazón de una forma no invasiva.

Con esta herramienta, una nueva rama de la medicina acababa de comenzar. El comportamiento del corazón había pasado de ser una incógnita a poder ser registrado y estudiado, provocando una revolución en el mundo científico. Este descubrimiento no sólo permitió que el ECG fuera visible, sino que además se descubrió que en el caso de sujetos sanos, el comportamiento de esta señal era totalmente predecible. Un profesional de la medicina correctamente instruido podía realizar diagnósticos de forma sencilla y acertada. Una vez definidos los patrones de las diversas dolencias cardiacas en la literatura científica se convirtió en algo sencillo para el mundo de la medicina.

Con la llegada de la computación, se empezó a trabajar con el procesado de este tipo de señales y con la llegada de los amplificadores, los DSP o los sistemas de adquisición, esta línea de investigación pegó un gran salto cualitativo. A pesar de dicho salto, estos sistemas aún dependían del factor humano en el sentido de que se necesitaba que un médico estuviera pendiente de los resultados y los interpretara. Este problema fue el que llevó a partir de los 80 a que diversos centros de investigación como el MIT, se concentraran en la búsqueda de algoritmos que permitieran un análisis automático de estas señales.

Durante esta década se desarrollaron algunos de los algoritmos más usados hoy en día para el análisis de las señales ECG. Estas investigaciones se toparon con un problema, la velocidad de los microprocesadores de la época no era suficiente para implementarlos en tiempo real por lo que las investigaciones quedaron en standby hasta que en los años 90 comenzaron a aparecer procesadores digitales con la suficiente potencia para llevar a cabo estos algoritmos y se empezó a probar de forma experimental lo que hasta hace unos años sólo podía probarse de forma teórica.

2.1.2. Señal electro-cardiográfica

El electrocardiograma (ECG) refleja la propagación de la despolarización y repolarización eléctrica de las diversas cámaras contráctiles del corazón. El término ECG hace referencia únicamente a cuando la captación de la señal se realiza con electrodos superficiales. El estudio del ciclo cardiaco se lleva a cabo utilizando el ECG como referencia temporal, de esta forma podemos dividirlo en dos partes, la primera de ellas estará asociada a la propagación de la excitación y recuperación de las aurículas, mientras que la otra, está asociada a la actividad ventricular.

Es esta característica de variabilidad temporal, la que ha hecho que los ECGs se hayan convertido en una herramienta tan importante a la hora de realizar diagnósticos o a la hora de abrir diferentes vías de investigación sobre las diferentes patologías cardiacas. La figura 2.1 muestra la forma que tiene este tipo de ondas en el dominio del tiempo:

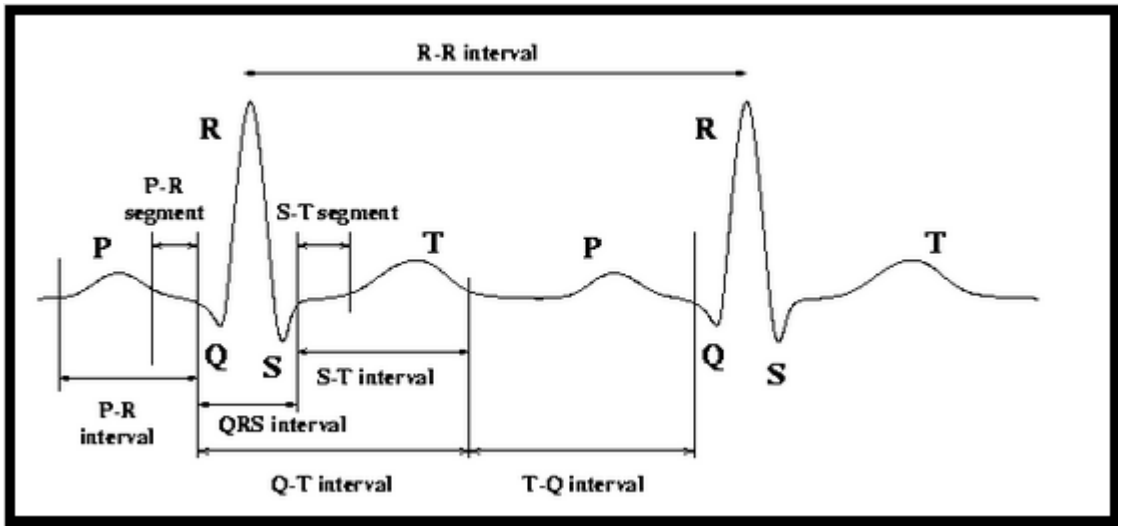

Figura 2.1. Definición de ondas, segmentos e intervalos en una señal ECG normal.

Observando la figura anterior lo primero que se debe hacer es conocer la diferencia entre segmento e intervalo. Se denomina segmento a la línea generalmente isoeléctrica que une una onda con otra sin incluir ninguna de ellas, por otro lado se denomina intervalo a la porción de ECG que incluye un segmento y una o más ondas.

Visto esto, a continuación se describen los tramos mostrados en la figura 2.1:

- Intervalo RR: Es la distancia entre dos ondas R consecutivas. Este intervalo es constante y su longitud dependerá de la frecuencia cardiaca. El cálculo de este tiempo será uno de los objetivos de este trabajo.
- Intervalo PR: Representa la despolarización auricular y el retraso fisiológico que sufre el estímulo a su paso por el nodo aurículoventricular (NAV). Se mide desde el inicio de la onda P hasta el inicio de la onda Q.
- Intervalo QRS: Mide el tiempo total de despolarización ventricular. Se mide desde comienzo de la onda Q o R hasta el final de la onda S. Es importante remarcar que el intervalo QRS incluye al conjunto de todas las ondas que conforman el complejo QRS, ya que tendrán un papel protagonista a lo largo de este trabajo puesto que serán las que se intentarán detectar a través del procesado de la señal.
- Intervalo QT: Este intervalo refleja la despolarización y la repolarización de los ventrículos, o lo que es lo mismo, la sístole ventricular.
- Segmento ST: Este segmento es el resultado de un periodo de inactividad entre la despolarización y el inicio de repolarización ventricular. Habitualmente es isoeléctrico y se extiende desde el final del QRS hasta el inicio de la onda T.

Una vez conocida la forma de onda que tiene un ECG, así como los segmentos e intervalos que la componen, se debe dejar claro que, esta forma de onda es el resultado de una serie de acontecimientos producidos en diversas etapas llevadas a cabo en el desarrollo de la actividad normal en cada latido del corazón.

En la figura 2.2 se ven cuáles son estas etapas y qué pasa en cada una de ellas

1. NSA (Sinus node activation): Este nodo es el responsable de comenzar con el latido normal del corazón. El NSA genera un impulso regularmente (de 60 a 100 veces por minuto), el cual se propaga a través de las aurículas causando la contracción de las mismas. Esta

contracción incrementa la presión provocando la apertura de las válvulas mitral y tricúspide y haciendo que de esta forma la sangre fluya hacia el ventrículo.

2. El impulso eléctrico alcanza el nodo auriculoventricular (NAV). Este nodo retiene el impulso durante un breve periodo de tiempo (100 ms aproximadamente) permitiendo la relajación de la aurícula mientras la sangre fluye hacia los ventrículos. Una vez que los ventrículos están llenos, su presión incrementa, cerrando las válvulas mitral y tricúspide. Una vez pasado estos 100 ms el impulso es propagado hacia el haz de His. Este nodo tiene una frecuencia de activación mínima de 50 pulsos por minuto así que en el caso de que el impulso no hubiera sido generado en el NSA el NAV regenerará por sí mismo la "cadena de latido".

3. El haz de His transmite el impulso desde la aurícula a los ventrículos.

4. Cuando el impulso pasa a través de las fibras de Purkinje esto provoca la contracción de los ventrículos y a medida que la presión aumenta, se abren las válvulas aortica y pulmonar. Al igual que pasaba con la NAV, las fibras de Purkinje tienen una frecuencia de activación mínima de 15 a 30 pulsaciones por minuto por lo que si el NSA o el NAV fallan esta será la frecuencia cardiaca del paciente.

5. Una vez que el ventrículo está vacío se producirá el cierre de las válvulas pulmonar y aórtica. Mientras que paralelamente se inicia de nuevo el proceso.

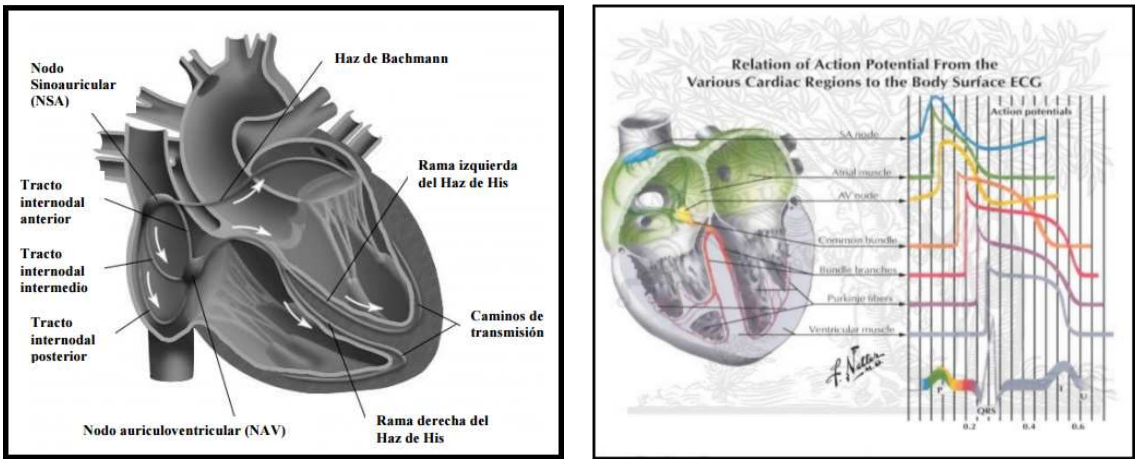

Figura 2.2. Estructura de conducción eléctrica del corazón y representación de la actividad eléctrica del corazón en diversas zonas [1].

2.1.3. Detección de complejo QRS

Una vez conocida, la forma de onda de un ECG y los intervalos que lo forman llegamos al punto clave que da sentido a la realización de este trabajo, la medida del intervalo RR, el cual se usará para conocer la frecuencia cardiaca de un sujeto. Como ya se explicó, el intervalo RR es aquel que se encuentra entre dos ondas R consecutivas, por tanto queda clara la necesidad de tener un buen algoritmo para la detección del complejo QRS.

Actualmente, podemos encontrar dos vertientes para la resolución de este problema, los algoritmos no-sintácticos y los sintácticos.

Los algoritmos sintácticos se basan en la aplicación de reglas sintácticas sobre la señal. Dichas reglas, se definen a partir de la morfología y secuencia de aparición de las ondas en el ECG, extrayendo de esta forma un patrón de la señal a detectar. Estos algoritmos son muy lentos, debido a la necesidad de la existencia de una correlación entre patrón y onda, por lo que no gozan de mucha popularidad.

Los no-sintácticos, al contrario, se basan en un pre-procesado de la señal, que elimina ruido y destaca las características de interés de la onda, comprobando la presencia de un QRS válido a partir de una regla de decisión. Esta regla de decisión se suelen basar en umbrales (ya sean adaptativos o no).

Los métodos no-sintácticos se utilizan con mayor frecuencia debido a que poseen una mayor velocidad de cálculo, lo que hace que sean idóneos para el cálculo en tiempo real. Un ejemplo de este método es el algoritmo creado en 1985 por Pan y Tompkins que a día de hoy se sigue usando y es la base de un gran número de investigaciones que han sido desarrolladas en estos últimos años para el procesado de señales ECG en tiempo real.

Algoritmo de detección QRS Pan & Tompkins

Como ya se ha explicado con anterioridad, este algoritmo fue creado en 1985 por los investigadores Jiapu Pan y Willis J. Tompkins [14] y a pesar del tiempo que ha pasado desde entonces sigue siendo hoy en día el algoritmo más popular en lo que se refiere a la detección en tiempo real del complejo QRS.

Existen varias características que hacen que este algoritmo se imponga al resto, sin embargo se podrían destacar como más importante los bajos requerimientos computacionales que tiene su etapa de pre-procesado. A continuación la figura 2.3 muestra el diagrama de bloques de esta etapa.

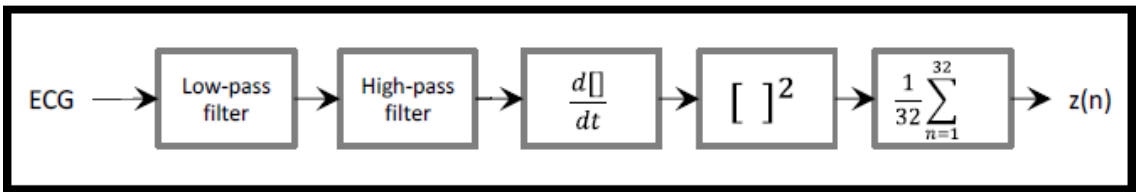

Figura 2.3.Etapa de preprocesado del algoritmo Pan & Tompkins [1].

Como se puede observar, en la figura 2.4, en la primera y segunda etapa, la señal (1) pasa por un filtro paso-bajo y otro paso-alto respectivamente, implementando de esta forma un filtro paso banda (2). Con este filtro se obtienen únicamente la banda donde se espera obtener el complejo QRS, eliminando otras señales que intervienen en la detección. El filtro paso bajo se encarga de la alta frecuencia y el paso alto elimina los componentes de continua y las ondas P y T.

Posteriormente esta señal ya filtrada se deriv. De esta forma son detectadas las pendientes pronunciadas tan características del complejo QRS.

Hasta este punto el procesado es lineal, sin embargo en este momento se eleva al cuadrado(3) las muestras y se realiza un promediado mediante una ventana de integración, cuyo ancho se elige de forma experimental (4). Con esta medida se consiguen dos cosas, la primera es tener toda la señal positiva debido a la etapa que eleva al cuadrado, así como hacer más pronunciadas las pendientes y la segunda está relacionada con el bloque integral. En esta etapa, al hacer el promediado se consigue eliminar las oscilaciones de poca duración que no corresponden al complejo QRS y se obtiene un pulso uniforme que se corresponde al complejo QRS.

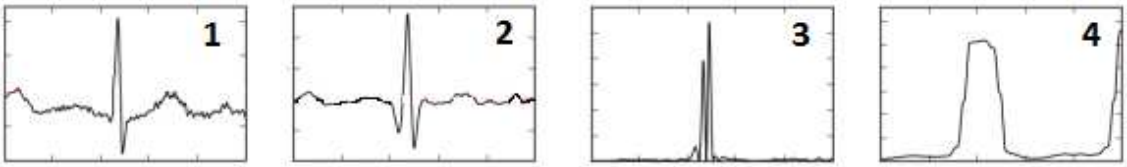

Figura 2.4.Evolución de la señal ECG [7].

2.2. Dispositivos FPGA

Las FPGA (Field Programmable Gate Array) son unos dispositivos semiconductores que contienen bloques de lógica cuya interconexión y funcionalidad puede ser configurada "in situ" mediante un lenguaje de descripción especializado (Verilog / VHDL). La lógica programable puede reproducir desde las funciones más simples, como por ejemplo las llevadas a cabo por una puerta lógica, hasta sistemas combinacionales más complejos.

Las FPGAs se utilizan en aplicaciones similares a los ASICs (dispositivos custom creados expresamente para desarrollar una función determinada), sin embargo son más lentas, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos. Sin embargo, las FPGAs tienen otras ventajas que las hacen ganar puntos frente a otros dispositivos similares, como la de ser reprogramables, característica que proporciona una gran flexibilidad en el flujo de diseño, sus costes de desarrollo y adquisición son menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

2.2.1. Antecedentes Históricos

Se conoce por el nombre de FPD (Field Programable Device) a todos los circuitos digitales utilizados para implementar hardware, donde el integrado tiene la capacidad de ser reprogramado por el usuario para la creación de distintos diseños.

Las FPLAs (Field Programmable Logic Array) o PLAs fueron los primeros dispositivos de este tipo que aparecieron. Estos, consistían en dos niveles de puertas lógicas (AND y OR) los cuales podían ser configurados y reprogramados dando lugar así a los dispositivos de hardware programable.

Con la aparición de los CPLDs (Complex Field Programable Device), este tipo de dispositivos dio un salto cualitativo, ya que permitían mejorar la velocidad a través de un array de varios PLDs colocados en bloque. En 1984 como evolución a estas CPLDs aparecieron las FPGAs.

Las FPGAs (Field Programmable Gate Array), en principio llamadas LCA (Logic Cell Array) fueron creadas en 1984 por los co-fundadores de Xilinx Ross Freeman y Bernard Von Derschmitt con una idea sencilla, la de crear una gate array tolerante a errores de diseño y reprogramables por el usuario.

Rápidamente adquirieron popularidad debido a su gran cantidad de ventajas como por ejemplo: Alta complejidad, bajo coste de desarrollo, fácil depuración, tolerancia a errores, tamaño reducido, alta fiabilidad, ...

El hecho de que este tipo de dispositivos sea reprogramable supone que tanto las interconexiones como las entradas y salidas también tienen que ser reprogramables. Para ello, este tipo de dispositivos disponen de una serie de elementos básicos configurables que se encuentran habitualmente posicionados en filas y columnas los cuales se conectan entre si y a los puertos de entrada y salida formando de esta manera lo que se podría denominar como una red de conexiones.

Existen tres tipos de celdas básicas en una FPGA, éstos son:

- CLBs (Configurable Logic Block): Son los bloques donde se implementa el diseño hardware.
- IOBs (In Out Block): Comunican señales internas del integrado con los pines físicos del dispositivo.
- Bloques de interconexión de celdas que permiten la conexión de los CLBs entre sí y con los IOBs.

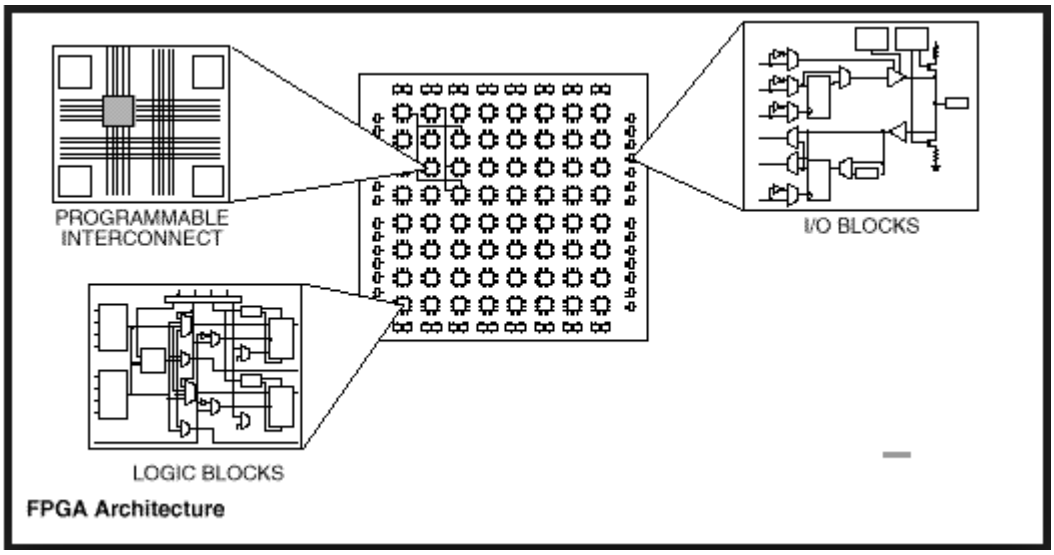

Figura 2.5.Estructura interna de una FPGA [8].

Este trabajo se llevará a cabo sobre una FPGA de Xilinx de la familia Virtex 5. En los siguientes puntos se realizará un análisis más detallado de los bloques mostrados en la figura 2.5. así como de otras características acerca de su arquitectura que serán cruciales para el desarrollo de este trabajo.

2.2.2. CLB: Configurable Logic Block

El CLB es el bloque principal para la implementación tanto de lógica secuencial como combinacional. Cada CLB está formado por 2 slices que no poseen conexiones directas entre ellos y están organizados en columnas. Estos slices poseen señales de acarreo de salida y entrada para la implementación de sumadores y multiplicadores. En la figura 2.6 se muestra una imagen que ilustra la estructura de una slice.

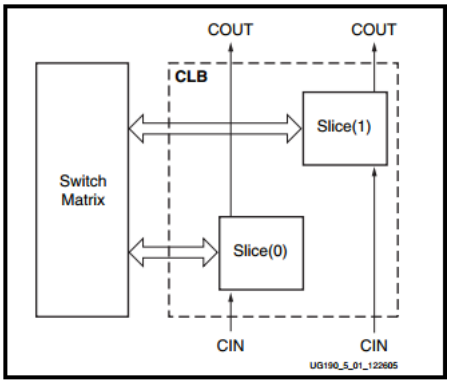

Figura 2.6.Estructura interna de una CLB [8].

En los CLB se pueden encontrar dos tipos de slice, los llamados SLICEL y SLICEM. Ambos poseen 4 LUTs (Look Up Table), cuatro registros, multiplexores y lógica de acarreo, con la diferencia de que los SLICEM además poseen recursos de memoria, para guardar datos mediante una RAM distribuida o para realizar desplazamientos como un registro de 32 bits.

En cada CLB se pueden encontrar 2 slices de los cuales sólo uno como máximo será un SLICEM, no existiendo nunca 2 SLICEM juntos. Sin embargo, si es posible encontrar 2 SLICEL en un mismo CLB.

En la figura 2.7 se comparan las estructuras de un SLICEM (a la izquierda) y un SLICEL (a la derecha):

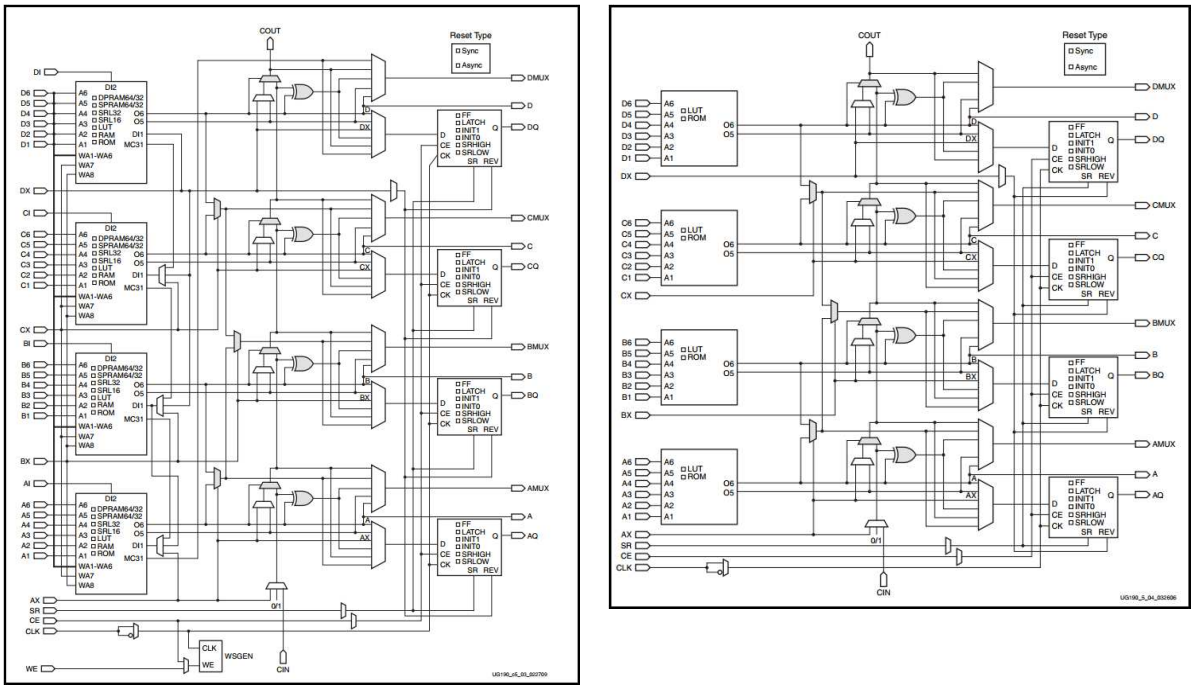

Figura 2.7.SLICEM vs SLICEL [8].

2.2.3. LUT: Look Up Table

Cada LUT es un recurso básico que permite implementar seis entradas y una salida o dos tablas de verdad de 5 entradas (compartiendo entradas entre ellas) y 2 salidas.

Si se deciden usar los multiplexores que se encuentran en cada slice a la salida de las 4 LUTs es posible llegar a implementar tablas de verdad de 7 u 8 entradas (dependiendo de si son usados 2 o 3 multiplexores). Se podrían

implementa tablas de verdad de más entradas, aunque para ello sería necesario disponer de otro slice más.

2.2.4. Elementos de almacenamiento

En un slice los elementos de almacenaje o biestables pueden ser configurados como flip-flops o como latches. Al configurarse como este último, el latch es transparente con el reloj a nivel bajo. Las señales de control, ya sean el reloj, el clock enable, el set/reset y el reverse son comunes para todos los elementos de un mismo slice.

2.2.5. RAM distribuida

Esta funcionalidad como ya se apuntó con anterioridad solo está disponible con los SLICEM. Este tipo de slice posee la funcionalidad de RAM síncrona distribuida y puede ser configurada de varias maneras implementando de esta forma diferentes configuraciones. En estas memorias, la escritura se efectúa de forma síncrona mientras que la lectura se realizará de forma asíncrona. La lectura se puede hacer también de forma síncrona utilizando flip-flops, sin embargo esto introducirá una cierta latencia.

2.2.6. Registros de desplazamiento

Estos registros también son exclusivos de los SLICEM. Es posible implementar un registro de desplazamiento de 32 bits sin usar los flip-flop del slice. De esta forma es posible introducir con LUTs latencias en series de datos de hasta 32 bits. También se pueden concatenar los LUTs del slice pudiendo conseguir retardos de hasta 128 ciclos de reloj.

2.2.7. DSP48E: DSP Slice

Las aplicaciones dedicadas al procesamiento de señales usan un gran número de multiplicadores y acumuladores, la mayoría de las veces implementados en unos slice específicos. En el caso de la familia de la Virtex 5 se encuentran a nuestra disposición varios de estos slices, dedicados y configurables para el desarrollo de este tipo de aplicaciones.

A continuación se muestra una imagen de la estructura de estos slice:

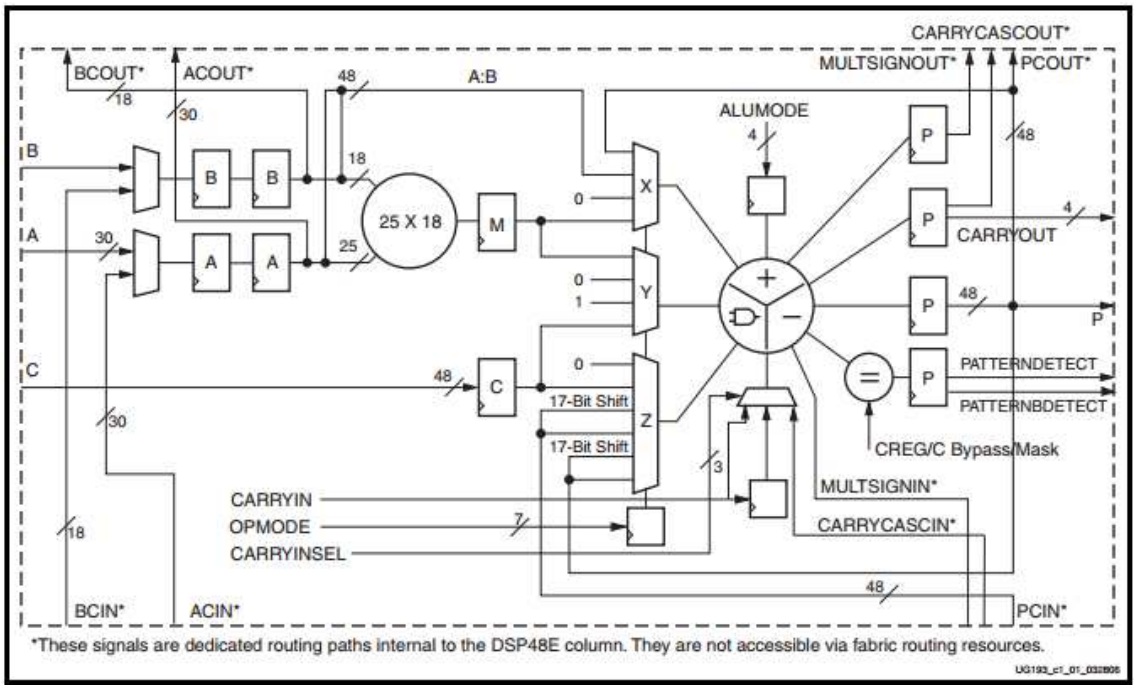

Figura 2.8. Slice DSP48E de la familia Virtex5 [8].

Como se puede observar cada uno de estos slices cuentan con un multiplicador 25x18, una unidad aritmético lógica (ALU) donde se pueden realizar operaciones lógicas, sumas y restas de hasta 48 bits, desplazamientos, comparadores y registros para la implementación de pipelines.

Cuenta con 3 entradas de datos (A de 30 bits, B de 18 bits y C de 48 bits) pudiendo tomarse A y B como entrada directa o en cascada, procedente de otro DSP48E así como de una salida directa P de 48 bits que también se puede concatenar con la ALU de otro DSP.

2.3. Objetivos

El objetivo global de este trabajo es el de realizar el diseño y la implementación de un sistema capaz de realizar de forma eficiente el procesado de una señal electro-cardiográfica (ECG). Para ello se superarán unos objetivos parciales que se podrían desglosar de la siguiente forma:

- Estudio de la señal biomédica ECG, comprender como se origina y qué ondas la componen con el fin de implementar una solución sobre FPGA que resuelva el tema de la detección del complejo QRS.
- Diseñar un camino de señal o datapath para la implementación del algoritmo en una FPGA de la familia Virtex 5.
- Comprobar el modelo del algoritmo en Matlab en coma fija y comprobar la pérdida de resolución obtenida con respecto al modelo realizado en coma flotante.
- Diseñar e implementar una solución eficiente y robusta que permita dar solución al problema de la detección del complejo QRS.

  Este objetivo se desglosará a su vez en dos partes. En la primera de ellas se realizará un bloque de procesado de la señal recogida por los sensores. En la segunda se planteará un algoritmo de detección.
- Comparar el algoritmo empleado con el algoritmo llevado a cabo por Pan y Tompkins (uno de los algoritmos más populares para la resolución de este tipo de problemas).

Capítulo 3. Algoritmo propuesto para la detección de complejos QRS.

En el presente capítulo se realizará una descripción del algoritmo que se empleará en este trabajo para la detección de complejos QRS. Se tratará de un algoritmo no sintáctico, en el que se pueden destacar dos etapas bien diferenciadas. En la primera de ellas se realizará un preprocesado de la señal. Esta etapa ayudará a eliminar el ruido y destacar las características de la onda que sean de interés. La segunda parte sin embargo, consiste en un algoritmo de detección en el cual se recibirá la señal procedente de la etapa de pre-procesado y se detectará la presencia de un QRS válido a partir de una regla de decisión definida por un umbral que variará de forma dinámica. A continuación se muestra el diagrama de bloques que ilustra a grandes rasgos las dos etapas anteriormente descritas.

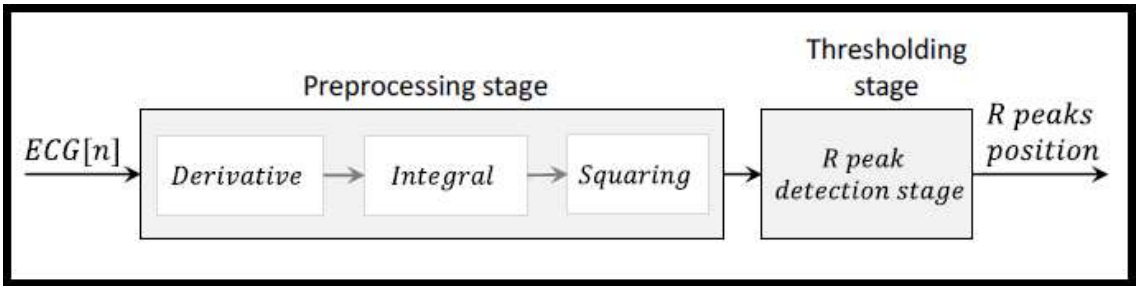

Figura 3.1. Algoritmo para detección QRS propuesto [1].

Como se puede observar, el algoritmo mostrado en la figura 3.1 posee ciertas similitudes con el algoritmo de Pan y Tompkins, sin embargo se puede ver que posee una etapa de preprocesado de menor exigencia en cuanto a lo que recursos se refiere lo que hará que sea una opción ideal para todas aquellas aplicaciones básicas que requieran conocer la posición del complejo QRS.

Debido a la importancia que posee la señal ECG para el correcto desarrollo del algoritmo, se trabajará tanto en las simulaciones y scripts realizados en Matlab como en las simulaciones funcionales y temporales realizadas con simulink con señales reales de pacientes.

La siguiente imagen representa parte de una de estas señales.

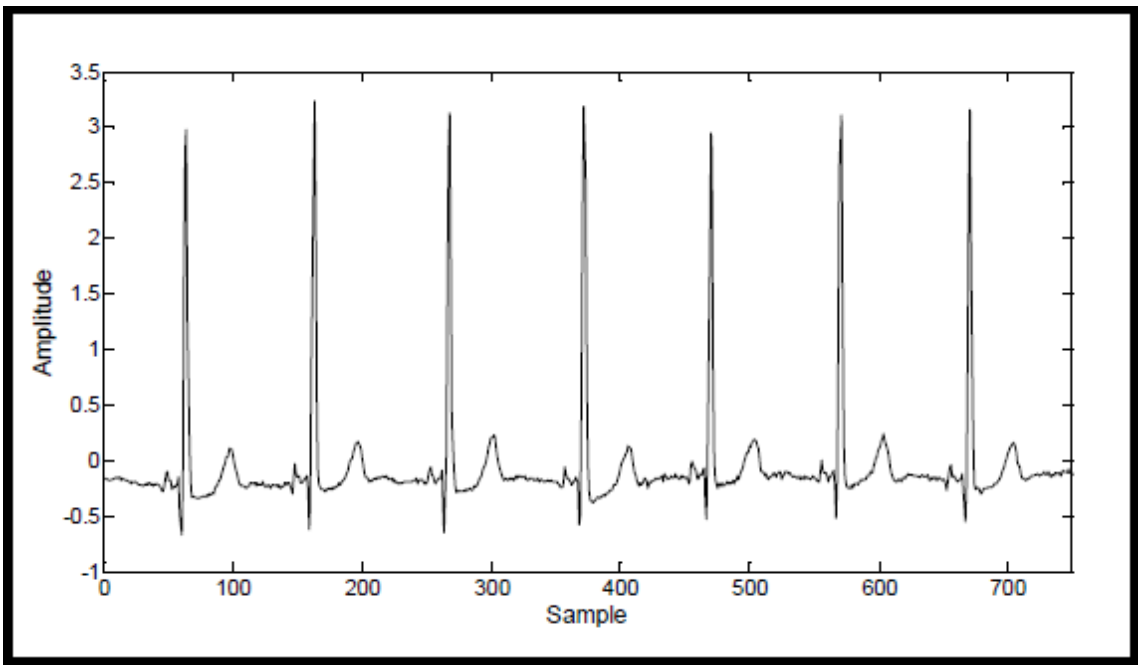

Figura 3.2. Ejemplo de señal ECG [1].

3.1. Descripción del algoritmo (coma flotante)

Como ya se ha explicado, el algoritmo consta de dos partes principales, una primera de pre-procesamiento y otra que se encargará de la detección QRS a través de la utilización de un umbral dinámico. Esta segunda etapa estará controlada por una máquina de estados. En las siguientes líneas se explicarán cada una de estas etapas de forma más detallada.

3.1.1. Etapa de preprocesado.

Esta etapa es de gran importancia ya que en ella las ondas de la señal ECG que carecen de importancia desaparecerán y se acentuarán las importantes.

Se pueden distinguir tres fases dentro de esta etapa. En la primera de ellas se realizará una derivada según el parámetro *Nder*, que será explicado más

adelante. Con esta derivada se detectarán las pendientes pronunciadas, las cuales son probablemente la característica principal del complejo QRS.

$$Y_0(n) = ECG(n) - ECG(n - Nder) \quad (1)$$

Posteriormente se realiza un promediado mediante una ventana de integración, cuyo tamaño viene marcado por el parámetro *N*, con el que se conseguirán eliminar las oscilaciones de poca duración que no corresponden al complejo QRS.

$$Y_1(n) = \frac{1}{N-1}\sum_{k=0}^{N-1} Y_0(n-k) \quad (2)$$

La última fase corresponde a la elevación al cuadrado de cada muestra. Con esto se hacen más pronunciadas las pendientes y se consigue una señal completamente positiva.

$$Y(n) = Y_1(n)^2 \quad (3)$$

La siguiente imagen muestra cómo va evolucionando la señal según va pasando por las diferentes fases de la etapa de preprocesado:

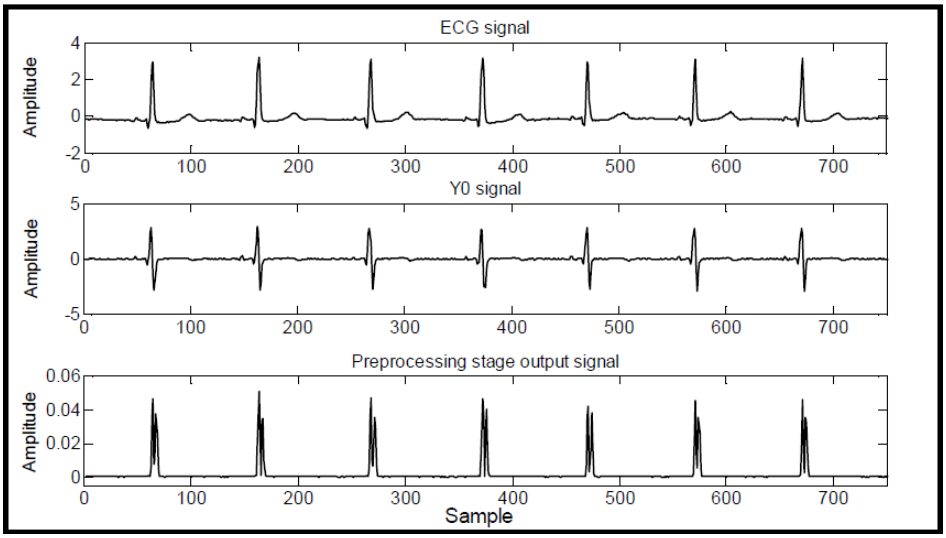

Figura 3.3 Evolución de señal ECG por las diferentes etapas del preprocesado [1].

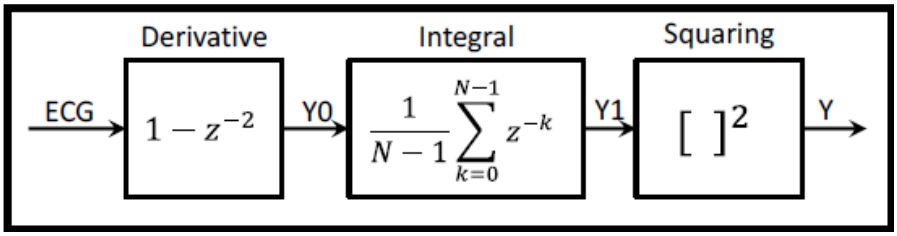

Figura 3.4 Etapa de preprocesado del algoritmo Pan & Tompkins [1].

3.1.2. Etapa de detección de complejos QRS

Como ya se ha comentado con anterioridad en este trabajo, esta etapa se llevará a cabo a través de un umbral dinámico que se usará como criterio en la decisión de si existe complejo QRS o no. El valor de este umbral estará marcado por una máquina de estados que irá modificándolo en función de los valores que se vayan obteniendo en la etapa previa.

Aunque este método es uno de los más populares para la detección del complejo QRS en sistemas de tiempo real, es posible que frente a una onda P con una amplitud demasiado grande, se obtenga un falso positivo o frente a un intervalo QRS de amplitud demasiado baja se obtenga un falso negativo. La cantidad de estos fallos obtenidos a la hora de simular el modelo en Matlab marcará si efectivamente es una solución útil o no.

Como ya se ha indicado esta etapa constará de una máquina de estados que será la que controle este umbral adaptativo. Esta máquina de estados constará de 3 estados principales que se describirán a continuación:

Estado 1: En este estado el algoritmo busca el máximo pico de la señal durante un intervalo igual al mínimo intervalo RR posible (RR_{min}) más la duración estándar del complejo QRS (QRS_{int}). El máximo valor se guardará como "R peak" o el valor máximo del complejo QRS. Es importante recordar que el tiempo entre estos picos es lo que marcaba la frecuencia cardiaca del sujeto.

Se pasará al estado siguiente cuando el tiempo RR_{min} + QRS_{int} haya terminado. Estos parámetros toman unos valores estándar que son 200 ms para RR_{min} y 60 ms para QRS_{int}.

Estado 2: La duración de este estado depende de la posición donde se encontrara el pico R en el estado anterior. De esta forma la máquina de estados permanecerá esperando hasta que pase un tiempo igual a RR_{min} menos el tiempo transcurrido entre la posición del último pico R y el final del estado 1. Haciendo esto, se garantiza que si existiera alguna onda P de gran amplitud, no se produzca un falso positivo.

Estado 3: Cuando el estado 2 termina, el nivel de threshold o umbral es la amplitud media de todos los picos R detectados. A lo largo de este estado su valor decrece con cada nueva muestra de la señal ECG con un factor que denominaremos *ParamTh*. El valor de este factor depende de la frecuencia de muestreo. Este estado termina cuando la señal ECG es mayor en amplitud que el valor de umbral. A continuación se muestra la máquina de estados propuesta.

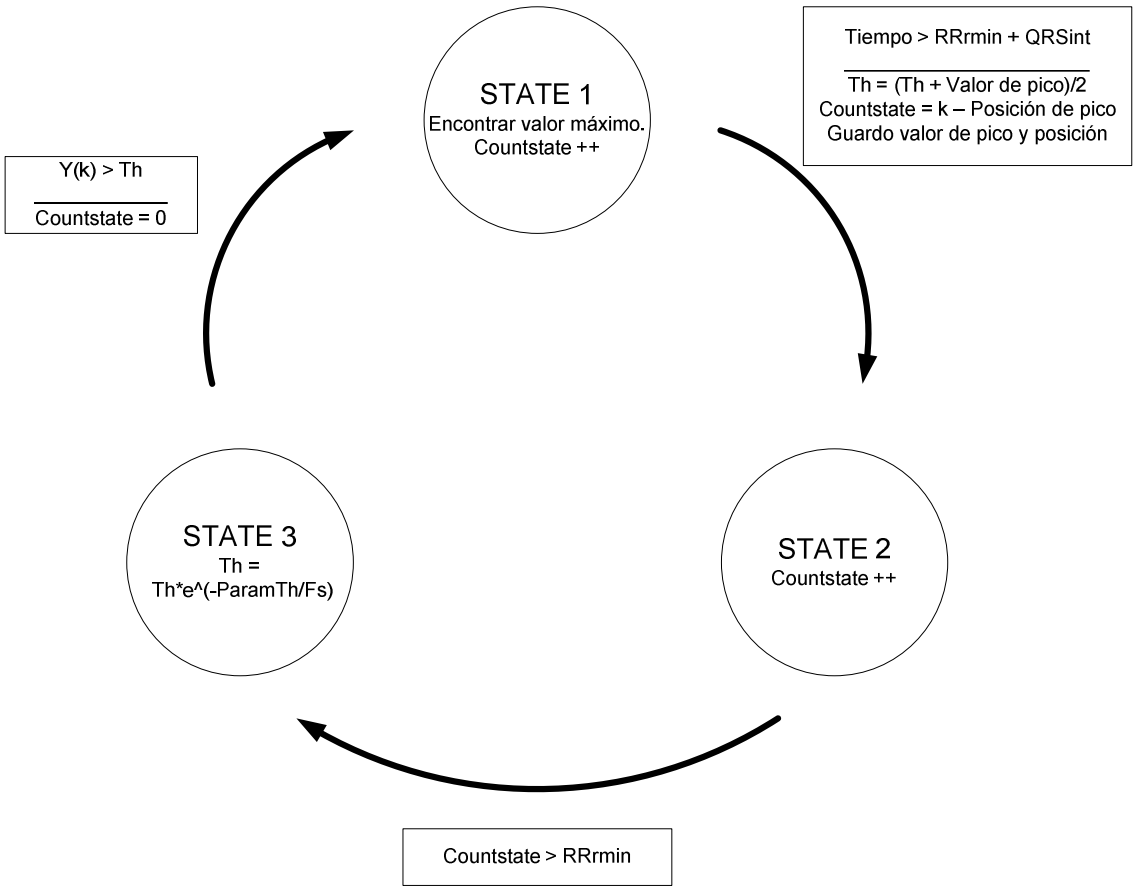

Figura 3.5 Máquina de estados del detector del complejo QRS.

La figura 3.6 muestra de forma gráfica sobre la señal ECG ya procesada como van transcurriendo los diferentes estados a medida que la señal va pasando por la máquina de estados.

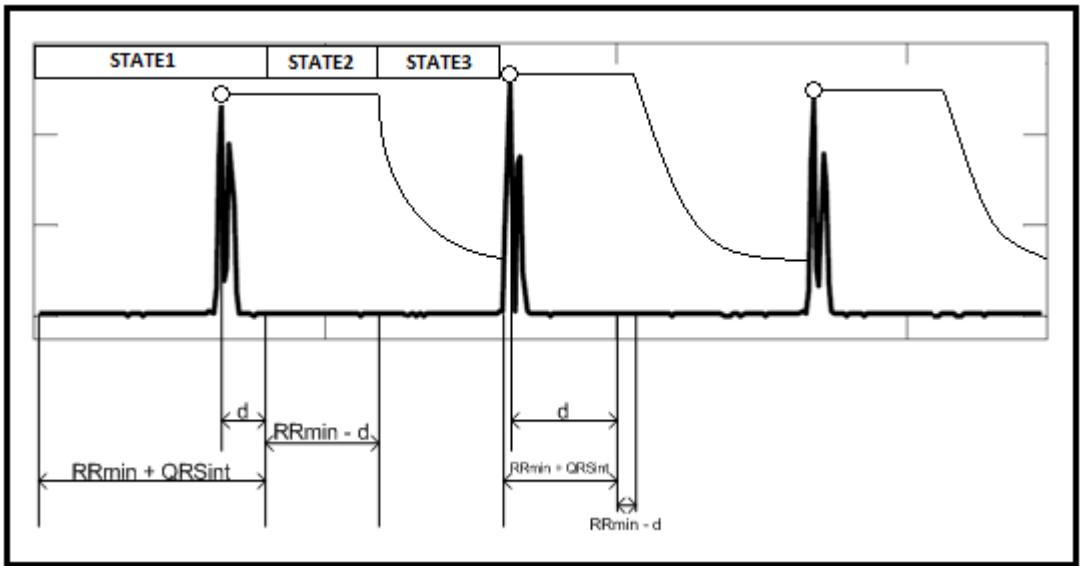

Figura 3.6 Correspondencia entre estados y señal ECG procesada.

3.1.3. Parámetros del algoritmo.

Como se ha podido ver en los dos puntos anteriores, el algoritmo planteado está basado en tres parámetros, el parámetro derivativo (*Nder*) y el tamaño de la ventana de integración (*N*) que se encuentra en la etapa de preprocesado, y el parámetro *ParamTh* (*PTh*) que se encuentra en la etapa de detección.

Para encontrar los valores óptimos según la frecuencia de muestreo (Fs) se utilizarán las siguientes formulas:

$$N = round\left(\frac{3 * Fs}{128}\right) \tag{4}$$

$$Nder = N - 1 \tag{5}$$

$$PTh = \left(\frac{0.7 * Fs}{128}\right) + 4.7 \tag{6}$$

Por lo que para diferentes frecuencias de muestreo se obtendrán los siguientes valores:

	N	*Nder*	*PTh*
128 Hz	3	2	5.4
256 Hz	6	5	6.1
360 Hz	8	7	6.8

Tabla 3.1. Valores de *N, Nder, y PTh* según *Fs*

3.2. Modelo del algoritmo en coma fija

Aunque la diferencia entre coma fija y coma flotante es más que de sobra conocida, a la hora de realizar un DSP, es necesario tener algunos aspectos bien claros. De esta forma, si tenemos un tamaño de palabra de 16 bits,. al usar una representación en coma fija poniendo la coma en el bit 15, se tendrán un gran número de decimales, pero muy pocas posiciones de enteros. Habrá por lo tanto mucha resolución pero poca diferencia entre los diferentes máximos y mínimos que podemos alcanzar. Si desplazamos la coma para obtener más enteros, pasará justamente lo contrario. Por lo tanto es necesario, mediante la posición de la coma, encontrar un compromiso entre los dos parámetros de medida más importantes de cualquier sistema: la resolución y el rango dinámico.

En un sistema en coma flotante sin embargo, la coma cambia automáticamente en función del número que se desea representar, de forma que siempre se alcanza el compromiso entre resolución y rango dinámico.

Aunque en general, la coma flotante facilita y optimiza la implementación de cualquier algoritmo y actualmente ya hay dispositivos FPGA que permiten su uso, su coste en cuanto a recursos, consumo y área de la pastilla requerida para su implementación hacen que a la hora de implementarlos se escoja el método de representación en coma fija, método que será empleado en el presente trabajo.

Llegados a este punto, y teniendo perfectamente definido en Matlab el algoritmo a implementar, es necesario convertir el modelo en coma flotante a coma fija, o lo que es lo mismo, realizar la representación numérica en coma fija para comprobar que la pérdida de resolución que se producirá como consecuencia de esta elección, no hace que nuestro algoritmo quede invalidado.

Para modelar el sistema en coma fija se utilizará la función *Quantize* de Matlab. Una vez hecho esto, procedemos a simularlos y comparar las salidas de ambos modelos para valorar el efecto de la pérdida de resolución

3.3. Simulaciones obtenidas en Matlab

En este punto se mostrarán las simulaciones de Matlab obtenidas para los modelos de coma fija y coma flotante. Se verá la perdida de resolución y se decidirá si el algoritmo expuesto es válido para su implementación en FPGA o no.

Lo primero que se hace con la señal ECG entrante es normalizarla. Para ello se divide entre 2048 ya que la señal proviene de un ADC de 12 bits. Posteriormente y haciendo uso de la función *Quantize*, se cambia la representación de la señal a punto fijo dejando 12 bits como parte entera y 0 como parte decimal.

```
32 -    ECG(:,1) = val(1,:)- base;
33      % ECG= ECG/2048;  % Normalizo
34      %Q= quantizer('fixed', 'saturate', 'floor', [12 11]);
35 -    Q= quantizer('fixed', 'saturate', 'floor', [12 0]);
36 -    ECG_fp= quantize (Q, ECG);
37
```

Figura 3.7 Captura de script de Matlab.

El formato para cambiar la representación numérica de una variable a punto fijo es el que se puede ver en la figura 3.7. La forma de realizarlo es igual para el resto de variables que han sido utilizadas en el modelo de punto fijo por lo que sólo se mostrará una vez, haciéndose extensible para el resto de casos.

A continuación se representarán superpuestas ambas entradas, así como su diferencia, obteniendo las siguientes gráficas:

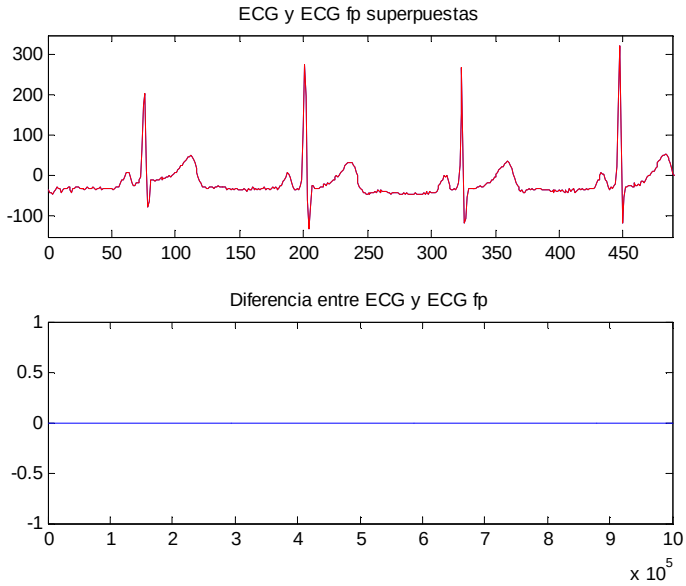

Figura 3.8 Entrada señal ECG.

Como ya se ha indicado anteriormente la señal proviene de un ADC de 12 bits, de esta forma, al haber realizado la representación en coma fija de la señal en 12 bits también (12 entero y 0 decimales), no se pierde ningún decimal en el proceso.

A continuación se procederá a ver el resultado obtenido tras el bloque derivativo.

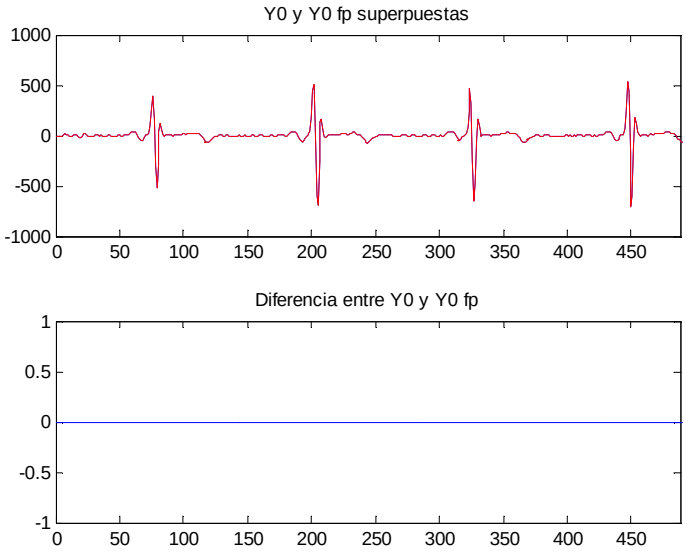

Figura 3.9 Salida del bloque derivativo.

En este punto el tamaño de palabra es de 13 bits, con 0 bits como parte decimal. Se puede observar que sigue sin haber errores de resolución.

A la salida del integrador sin embargo al existir una división, existirá truncamiento de datos, esto provocará una pérdida de resolución tal y como se puede ver en la siguiente gráfica, el tamaño de palabra en este punto es de 34 bits con 17 bits decimales.

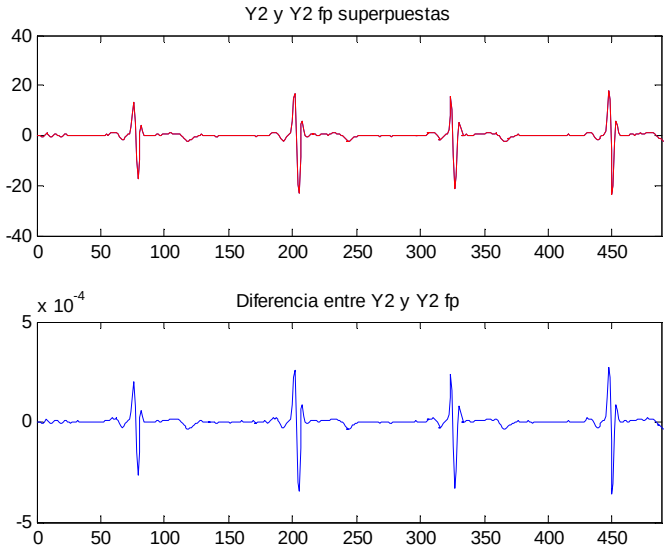

Figura 3.10 Salida del bloque integrador.

Y ya por último estaría la salida del bloque multiplicador obteniendo:

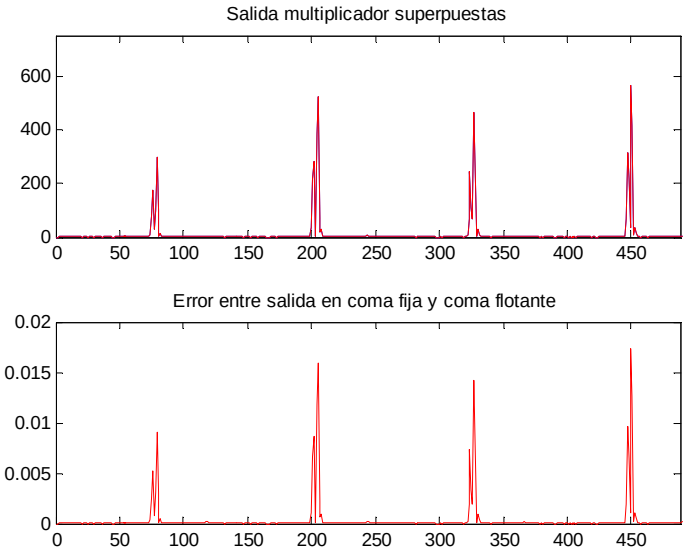

Figura 3.11 Salida del bloque multiplicador.

Como se puede observar a la salida de este último bloque cuyo tamaño de palabra es de 68 bits de los cuales 34 son decimales, el error arrastrado de la multiplicación del bloque anterior se mantiene, sin embargo, es de un orden de magnitud de 10^{-9} por lo que no supone ningún problema.

Llegados a este punto, la señal ECG ya procesada entra a la máquina de estados. En la máquina de estados se trunca el dato quedándonos con un tamaño de palabra de 16 bits de los cuales se tienen 0 bits decimales. Esto se hace para tener un dato más manejable a la hora de implementar el algoritmo en una FPGA. El parámetro de threshold se pasa también a un tamaño de palabra de 16 bits con 0 bits como parte decimal.

De esta forma se obtienen los siguientes resultados:

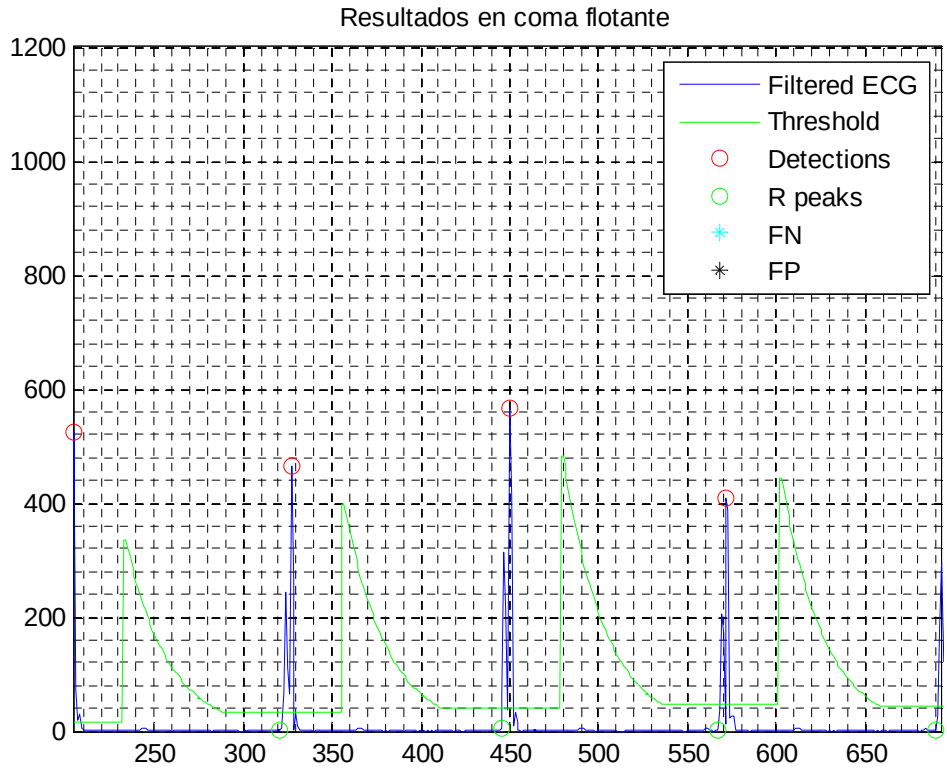

Figura 3.12 Gráfica de resultados de la detección de picos en coma flotante.

```
*** Paciente 16272 coma flotante***
  Total peaks: 7988
      TP: 7985
      FN: 3
      FP: 9
        * Se: 99.962444
        * P+: 99.887416
```

Figura 3.13 Resultados de la detección de picos en coma flotante.

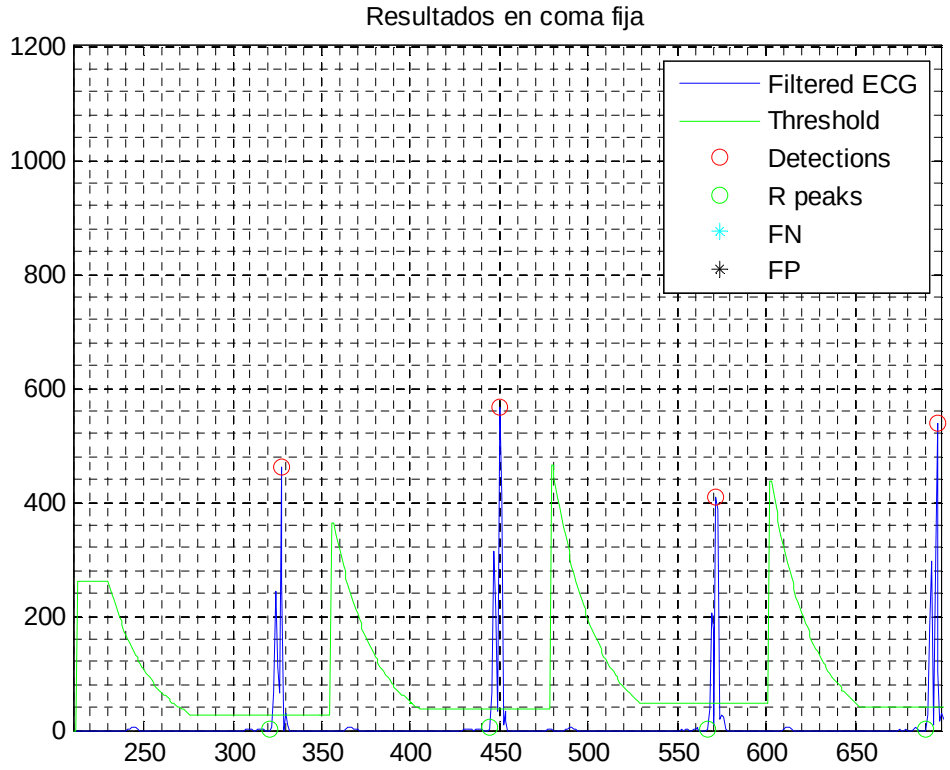

Figura 3.14 Gráfica de resultado de la detección de picos en coma fija.

```
*** Paciente 16272 coma fija***
  Total peaks: 7988
      TP: 7985
      FN: 3
      FP: 15
        * Se: 99.962444
        * P+: 99.812500
```

Figura 3.15 Resultado de la detección de picos en coma fija.

Atendiendo a los resultados obtenidos, y sabiedo que:

- TP: True Positive, este valor hace referencia a los picos correctamente detectados
- FN: False Negative: este valor hace referencia a los picos no detectados.

- FP: False positive, este valor hace referencia a las muestras detectadas como picos sin serlo.
- Se: Sensitivity, este valor hace referencia al porcentaje de picos correctamente detectados.
- +P: Positive predictivity, este valor hace referencia a la capacidad del detector de discriminar los picos R frente a los que no lo son.

Se puede observar cómo han aumentado de forma poco significativa los falsos positivos. Esto hará que el porcentaje de predectividad positiva disminuya, lo que significa que en coma fija el sistema discrimina un poco peor los picos R de otras anomalías que se puedan causar. Sin embargo se puede ver que la sensibilidad se mantiene, lo que significa que el sistema sigue detectando de forma correcta los picos R existentes.

Con estos datos podemos concluir que el sistema aunque ha empeorado ligeramente con respecto al modelo en coma flotante, sigue comportándose de forma correcta con unos porcentajes de fiabilidad elevados.

Capítulo 4. Implementación del algoritmo

Una vez comprobado que el algoritmo diseñado es válido, llega el momento de implementarlo sobre la FPGA.

Como ya se comentó en el capítulo de antecedentes, para llevar a cabo este trabajo se ha elegido la familia de la Virtex5 de Xilinx.

A lo largo de este capitulo se tratará de exponer la arquitectura del sistema, el diagrama de bloques y la elección del datapath y se mostrarán las diferentes simulaciones funcionales obtenidas a lo largo de todo el proceso de desarrollo.

4.1. Diagrama de bloques

Aunque como ya se explicó anteriormente, el algoritmo modelado en Matlab constaba de dos etapas principales correspondientes al pre-procesado y a la detección de complejos QRS, a la hora de implementar el sistema, se ha decidido incluir una etapa nueva denominada etapa de configuración. Esta etapa se encargará de detectar un cambio en la frecuencia de muestreo y enviar a cada bloque los parámetros correspondientes para esa frecuencia en particular.

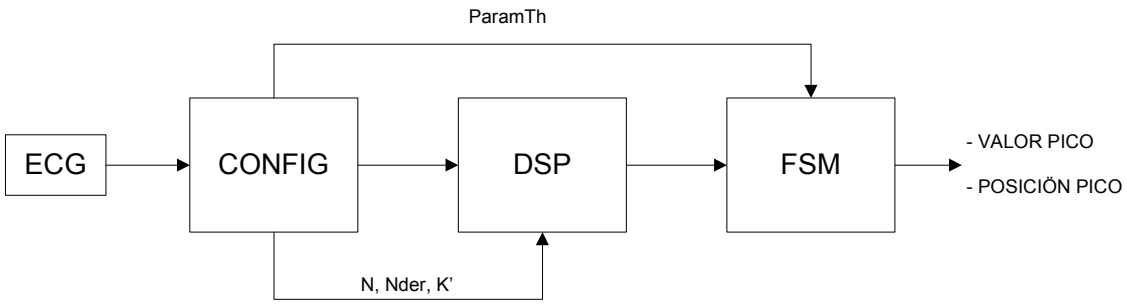

Figura 4.1 Diagrama de bloques general.

A continuación se describirá cada uno de estos bloques mostrados la figura 4.1 en detalle.

4.1.1. Bloque de Configuración.

Este bloque es el primero por el que pasa la señal y su función es la de detectar cambios en la frecuencia de muestreo y cambiar de forma dinámica los valores de los parámetros del algoritmo según sea una u otra.

La entidad que forma este bloque está representada por la siguiente figura:

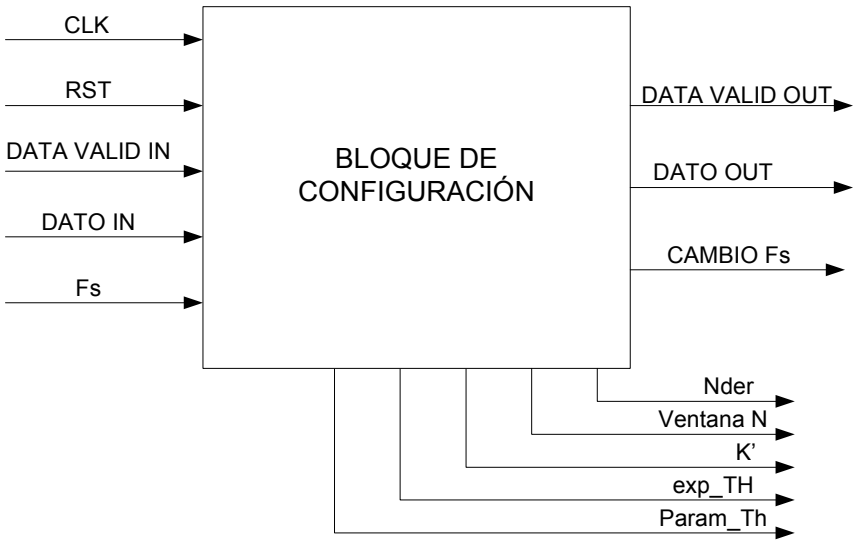

Figura 4.2 Entidad del bloque de configuración.

Como se puede observar este bloque tiene por entradas:

- CLK: Reloj del sistema funcionando a 50 MH
- RST: Reset asíncrono del sistema.
- Dato in: dato de 12 bits proveniente del ADC.
- Data valid in: Señal de un bit proveniente del ADC que se pone a '1' durante un clk cuando existe un dato válido.
- Fs: Frecuencia de muestreo. Esta señal consta de 3 bits. si el de menor peso está a '1' la Fs será igual a 128 Hz, si el bit 1 está a '1' la Fs será de 256 Hz y si el bit de mayor peso es '1' la Fs será de 360 Hz.

En cuanto a las salidas se encuentran las siguientes:

- Dato out: dato de 12 bits hacia el DSP.
- Data valid out: Señal de un bit que se pone a '1' durante un ciclo de reloj cuando existe un dato válido saliente hacia el DSP.
- Cambio Fs: Señal de un bit que se pone a '1' durante un ciclo de reloj cuando se detecta un cambio de Fs. Este bit se propagará por el sistema junto con el data valid asociado a una muestra determinada y se evaluará a la entrada de cada bloque, de tal forma que si se hubiera producido un cambio de Fs, antes de que la muestra afectada entre a dicho bloque, este ya habrá actualizado sus valores..
- El resto de señales de salida los parámetros necesarios para llevar a cabo el algoritmo. Estos parámetros estarán almacenados en el interior del bloque y dependiendo de la Fs que esté en cada momento se seleccionará a través de un multiplexor uno u otro.

A continuación se muestra un diagrama de la arquitectura interna de este bloque:

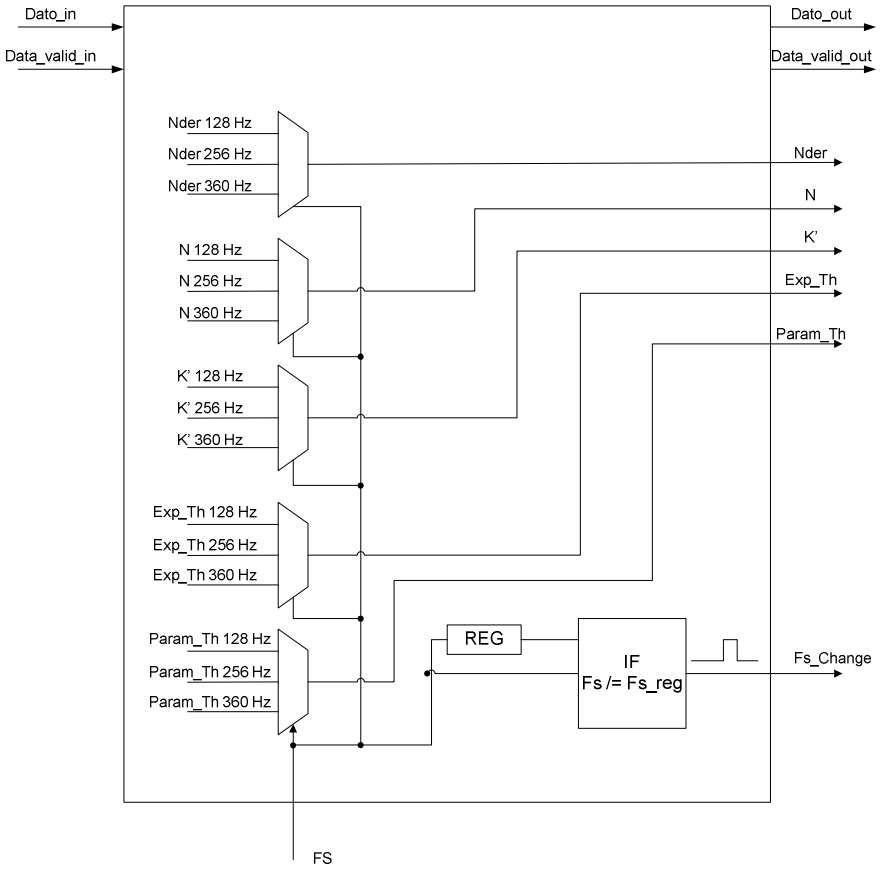

Figura 4.3 Arquitectura interna del bloque configuración.

4.1.2. Bloque DSP.

Este bloque es donde se realiza el preprocesado de la señal ECG tal y como ya se comentó en el análisis del algoritmo propuesto realizado en el punto 3.1 de este trabajo.

La entidad que forma este bloque es la que se muestra a continuación:

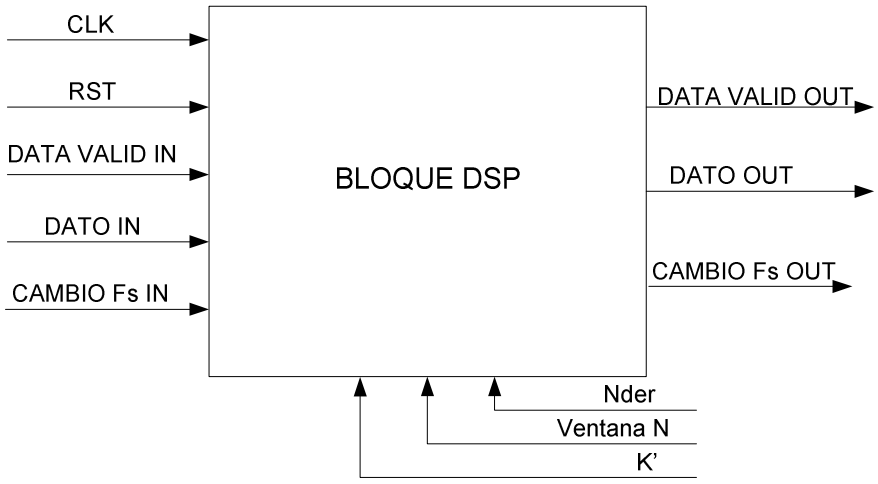

Figura 4.4 Entidad del bloque DSP.

Internamente este bloque DSP se divide en otras 3 partes:

Figura 4.5 División del bloque DSP en subpartes.

Como se puede observar en la figura 4.5, entre bloques, se han incluido pipelines con una doble función. La primera es la de segmentar la lógica combinacional de cada bloque para evitar los fallos producidos por el incumplimiento de tiempos, la segunda consiste en evaluar la señal de cambio de frecuencia de muestreo, haciendo que si esta cambia, se retarde el dato y se permita actualizar los parámetros que afectan al siguiente bloque.

Después de dejar claro esto, se pasará a describir la arquitectura empleada en cada una de las subpartes.

PRIMERA FASE DSP: Fase derivativa.

La arquitectura empleada para la realización de esta parte, es la mostrada a continuación:

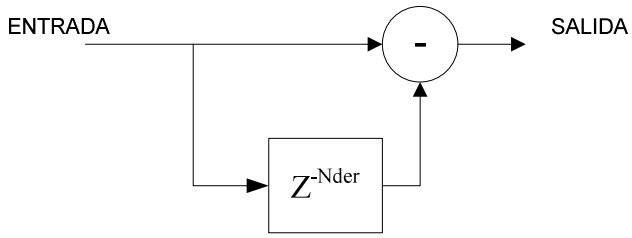

Figura 4.6 Arquitectura primera parte DSP.

Se trata de un registro de desplazamiento que retarda el dato un factor *Nder* y se lo resta al dato actual. El registro de desplazamiento se ha implementado aprovechando uno de los recursos de la familia Virtex5, los registros de desplazamiento SRL16E.

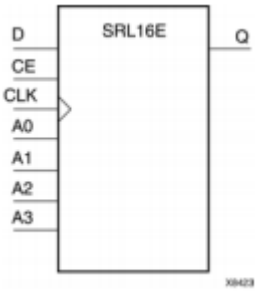

Figura 4.7 Registro de desplazamiento SRL16E.

Estos registros son unos elementos de diseño que se encuentran en las LUT. Las entradas A0, A1, A2 y A3 permiten elegir la longitud del registro. Estas entradas pueden configurarse como fijas o como es nuestro caso, ajustarlas dinámicamente según marque la frecuencia de muestreo. El valor "0000" marcará un tamaño de 1 bit de desplazamiento, mientras que con "1111" éste será de 16 bits.

Para poder implementar un registro de desplazamiento para un dato de 12 bits, se generó un array de estas primitivas haciendo uso de la sentencia for .. generate.

SEGUNDA FASE DSP: Fase Integral.

La arquitectura para llevar a cabo esta parte es la que se muestra a continuación:

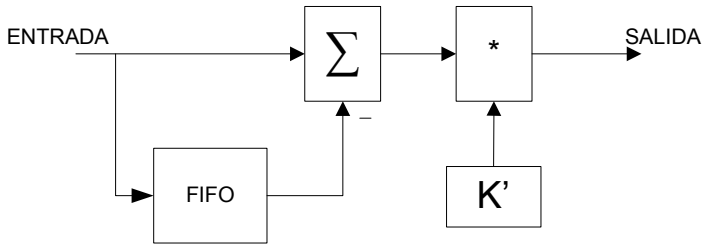

Figura 4.8 Arquitectura segunda parte DSP.

En este caso, como se puede apreciar en la figura 4.8, la arquitectura consiste en un acumulador seguido de un multiplicador.

El acumulador hace la suma de los datos según marca el parámetro $N_{windows}$ o ventana de integración. Una vez alcanzado ese número de datos, se resta al dato nuevo el primero que entró (almacenado en la FIFO) y el resultado se suma al total. De esta forma, se conseguirá tener siempre una suma de n valores.

Por otro lado, para obtener la media de esa suma calculada en la etapa anterior, es necesario dividir por el número de datos almacenados en el acumulador, o lo que es lo mismo, dividir entre *N*. Debido a que computacionalmente una división es muy costosa a nivel de recursos en una FPGA, se decidió multiplicar el resultado por la inversa de *N*, de esta forma:

$$K' = \frac{1}{N} \tag{7}$$

De esta forma se evitará hacer la división, llegando al mismo resultado.

Esta división se representará en 17 bits, el hecho de que el resultado de la operación tenga infinitos decimales, hace que al truncarlo se introduzcan errores por pérdida de resolución. Este tema se vio en el modelo en coma fija realizado en Matlab, y después de valorarlo, se llegó a la conclusión de que se podía obviar esta pérdida.

Para realizar esta multiplicación se ha utilizado uno de los slices DSP48E que ya fueron descritos en el capítulo 2.2. La configuración utilizada en este slice se muestra en la siguiente figura:

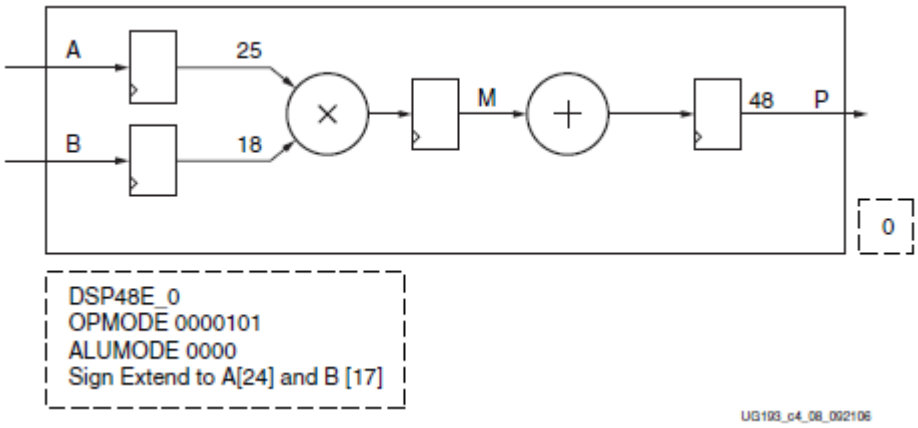

Figura 4.9 Configuración DSP48E [8].

Cabe destacar que la extensión del signo cuando se trabaja con datos menores de 25 x 18 bits (como es nuestro caso) es muy importante ya que el no hacerlo supone un resultado erróneo.

TERCERA FASE DSP: Cuadrado de la señal.

La arquitectura empleada es este último bloque del DSP es la siguiente:

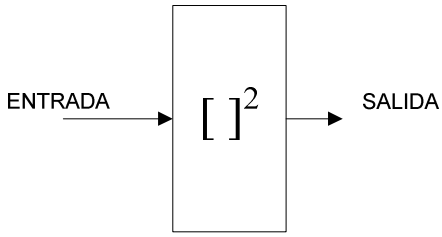

Figura 4.10 Arquitectura tercera parte DSP

Como se puede intuir, esta parte constará de un multiplicador que eleve al cuadrado el valor de la señal en este punto. Esto se llevará a cabo utilizando como en el bloque anterior el slice DSP48E. Sin embargo, existe una diferencia importante con respecto a la configuración empleada en el punto anterior. El tamaño de palabra del dato en este punto es de 34 bits, por lo que es necesario actuar con varios DSP48E en cascada, más concretamente con cuatro.

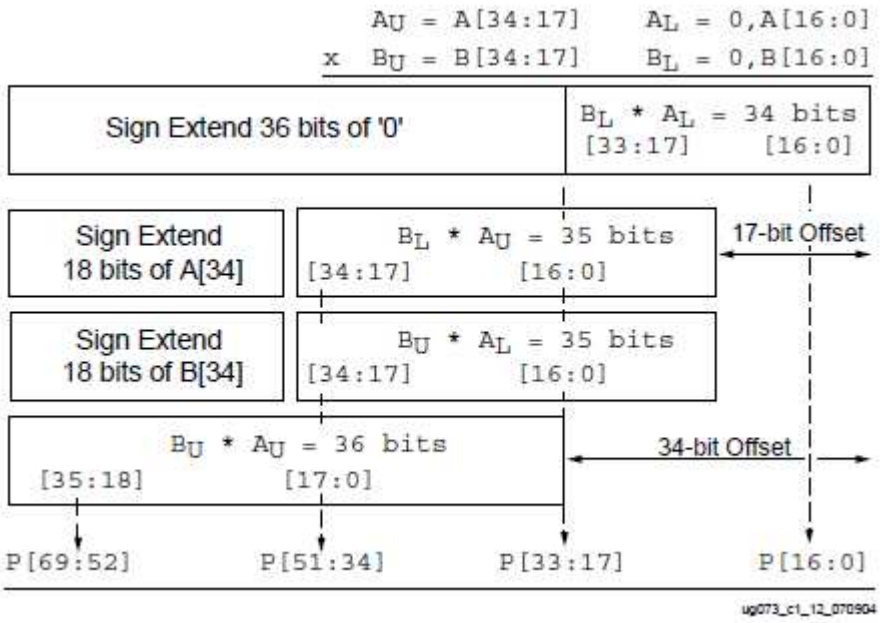

Figura 4.11 Multiplicador 35 x 35 a partir de multiplicadores 18 x 18 [9].

La figura 4.11 muestra en detalle cómo se crea un multiplicador de 35 x 35 bits a partir de multiplicadores 18 x 18 bits. La notación "0, A[16:0]" indica que se ha puesto un cero seguido de 17 bits (los 17 bits menos significativos de A) para formar un número positivo en complemento a 2.

Al separar dos números en complemento a 2 en dos partes, sólo la parte correspondiente a los bit de mayor peso debe llevar el bit de signo original. La parte correspondiente a los bits menos significativos debe tener su bit de signo forzado a '0' para convertirlo en un operando positivo.

Aunque pueda parecer lógico separar un número positivo en la suma de dos número positivos, cuando se habla de números negativos la cosa ya no está tan clara. Después de la separación, la parte más significativa es más negativa de lo que lo era antes, por lo que hace falta sumarle la parte correspondiente a los bits menos significativos que le fueron sustraídos.

Los Slice DSP48E con multiplicadores 18 x18 y su bloque ALU configurado como sumador, pueden usarse para realizar la suma de los 4 productos parciales que se muestran en la figura 4.11. Los productos parciales menos significativos, deben ser desplazados 17 posiciones antes de ser sumados con el siguiente producto parcial más significativo. Esto se lleva a cabo con la

entrada al DSP PCIN y seleccionando ésta como entrada del multiplexor Z. El proceso completo de multiplicación, desplazamientos, y sumas, usan un sumador en cascada para formar el resultado máximo de 70 bits que puede alcanzar esta configuración de DSP. La siguiente imagen ilustra esta construcción:

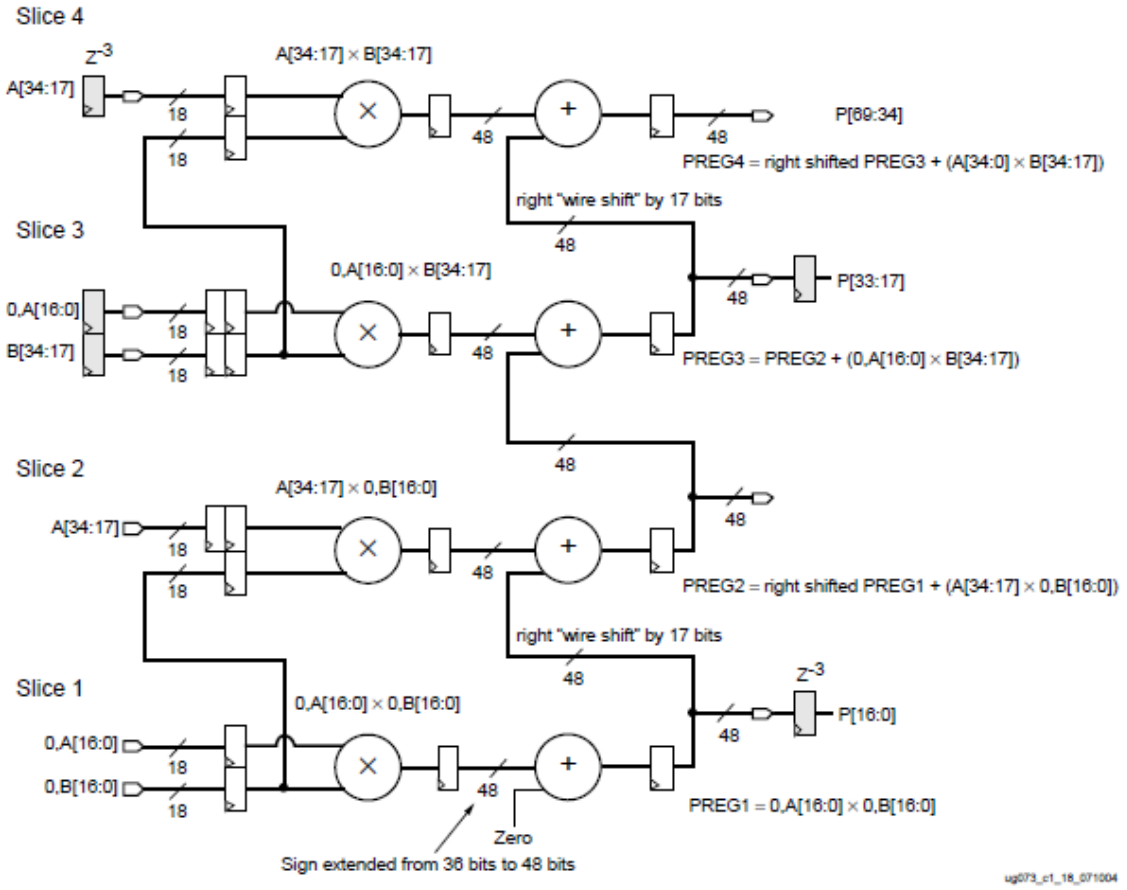

Figura 4.12 Multiplicador 35 x 35 [9].

4.1.3. Bloque FSM

Éste es el último bloque del algoritmo. La máquina de estados será la encargada de la detección del pico R permitiendo almacenar su posición y su valor. La entidad de este bloque es la mostrada a continuación:

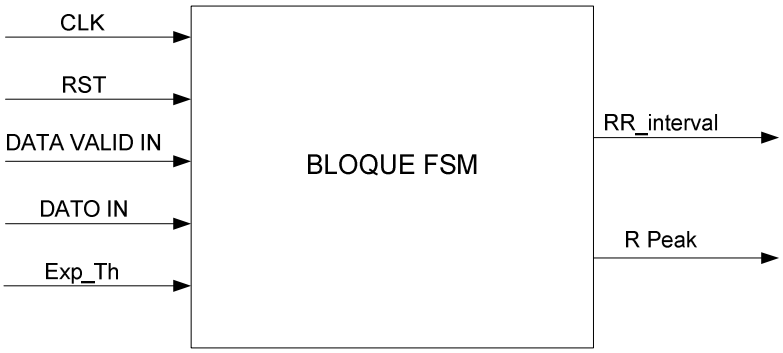

Figura 4.13 Entidad del Bloque FSM

A este bloque irán entrando las señales ya procesadas para ser evaluadas en busca de los picos R del ECG. Se debe remarcar la aparición del parámetro de entrada *exp_th*. Este parámetro sustituye a *ParamTh* y se calcula de la siguiente forma:

$$exp\_TH = e^{-\frac{ParamTh}{Fs}} \quad (8)$$

El resultado de esta operación para las tres frecuencias de muestreo que se manejan (128, 256 y 360 Hz), se almacena en el bloque de configuración y se proporciona a la máquina de estados según el valor de *Fs.* Esto nos permite evitar hacer el cálculo expresado en (8), el cual es muy costoso en cuanto a lo que recursos de la pastilla se refiere.

Es importante apuntar, aunque se verá más adelante, que este valor no se actualiza directamente, debido a que esto podría corromper el pico encontrado en ese ciclo de la máquina de estados. A continuación se mostrará el diagrama de bloques de la máquina de estados implementada así como las condiciones para las transiciones:

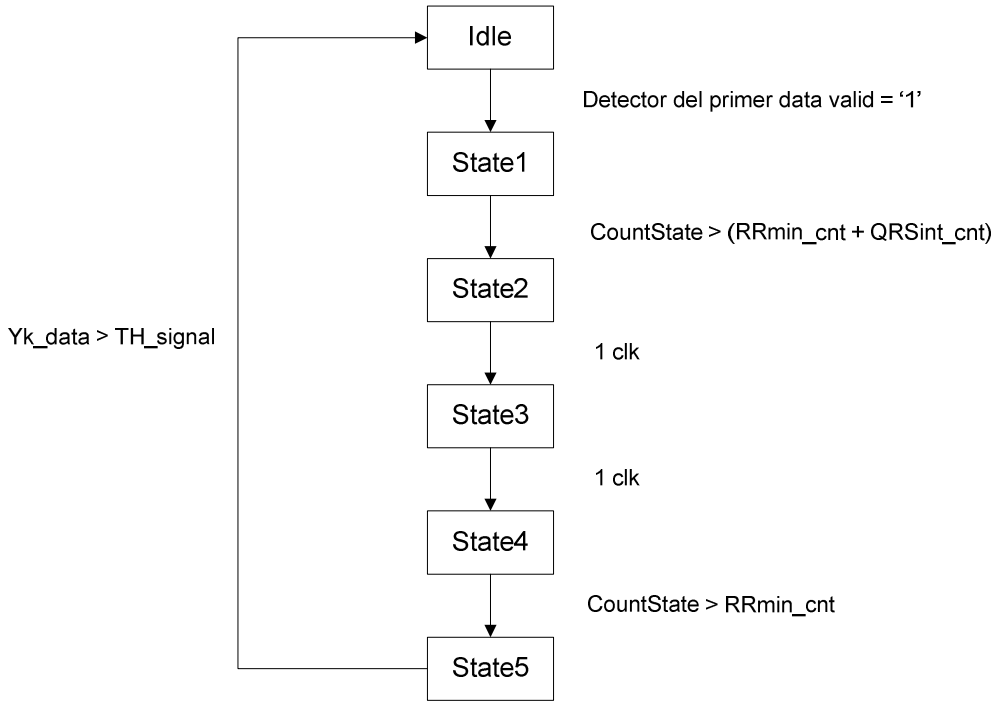

Figura 4.14 Máquina de estados y transiciones

Lo primero que se puede ver es que la máquina de estados finalmente implementada es ligeramente diferente a la descrita en capítulos anteriores. Se pueden encontrar diversas diferencias con respecto a la implementada en Matlab, la más notable es la aparición de 3 estados (de transición) nuevos.

A continuación se describirá brevemente las acciones tomadas en cada estado.

- Estado Idle: La máquina de estados comienza en un estado de reposo, donde permanece hasta que recibe el primer data valid. En este momento levanta un flag que se mantendrá a nivel alto hasta que se reinicie el sistema, una vez que la primera muestra a llegado a la máquina de estados´, este estado de reposo se convertirá en un estado de transición.

  Este estado representa el inicio de una nueva transición, y como tal es aquí donde se evaluará si ha habido o no un cambio de frecuencia de muestreo para la actualización de *exp_TH.*
- Estado 1: En este estado se realiza una cuenta para completar el tiempo definido por la suma de RR_{min} y QRS_{int}. Este tiempo será controlado con la realización de una cuenta, habiendose realizado previamente los cálculos necesarios para encontrar el valor de cuenta necesaria que

cumpla con los 200 ms que dura *RRmin_cnt* mas los 60ms que dura *QRSint_cnt*. Para realizar estos cálculos, es imprescindible conocer la frecuencia de funcionamiento del sistema, la cual en el caso de este trabajo se fija en 50 MHz. Cada ciclo de reloj por lo tanto dura 20 ns. Con este valor ya se puede crear un contador, al que se ha denominado *CountState* que contará tantas veces como sea necesario para llegar a los 200 ms marcados como límite El resultado puede ser calculado con una simple regla de 3:

$$limite\ de\ cuenta = 200\ ms * \frac{1\ clk}{20ns} \tag{9}$$

En el estado 1 también se realiza la búsqueda del Pico R. Para realizar esta tarea se ha utilizado el criterio de la primera derivada de la señal ECG. Este indica que cuando una función f'(x) cambia de positivo a negativo en un punto c, entonces f tiene un máximo relativo en (c, f(c)).

De esta forma se ha diseñado una lógica que realiza esta tarea.

El diagrama de bloques de este detector de máximo locales es el mostrado a continuación.

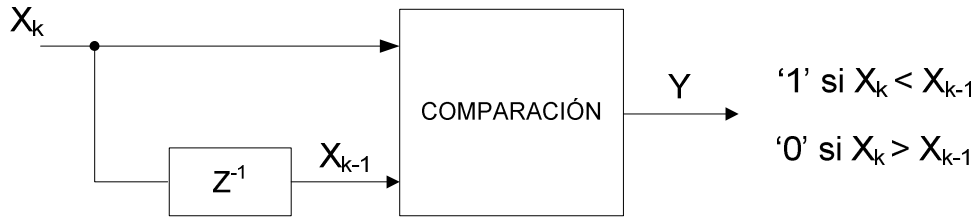

Figura 4.15 Detección de máximos

Este sistema proporciona un nivel alto en la señal *Y* siempre que exista un descenso, el máximo se encontrará en la muestra que provoca esta subida de nivel. De esta forma se almacenará el dato siempre que *Y* sea '1' y X_k sea mayor que el dato almacenado. Para realizar esta función se ha diseñado una lógica como la que se muestra a continuación.

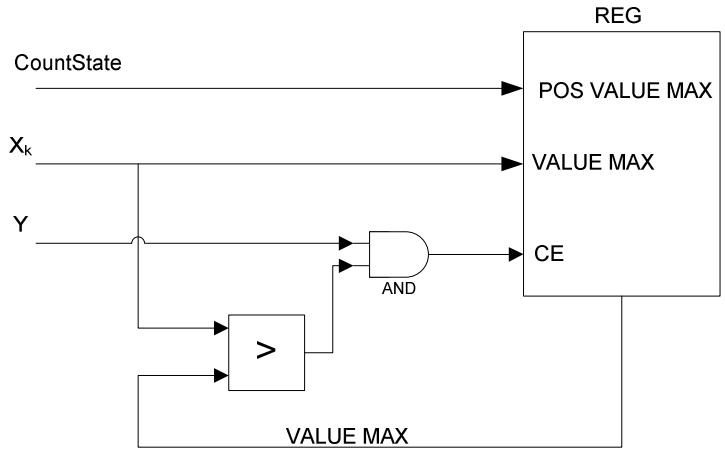

Figura 4.16 Almacenamiento Valor y Posición de pico

- Estado 2: En este estado, se obtiene el valor de inicio de cuenta del contador *CountState* que se requerirá en el estado 4

  También se almacenará los valores de posición y máximo obtenidos en el estado anterior.
- Estado 3: En este estado se actualiza el valor del umbral o Threshold. Para ello es necesario el valor del máximo almacenado en el estado anterior y el valor del Threshold antiguo. En el inicio de la máquina de estados, el valor que se le da a este parámetro es 0.

$$Th = \frac{Th + valor\ de\ pico}{2} \tag{9}$$

- Estado 4: En este estado y partiendo del valor de Countstate obtenido en el estado 2, se inicia una cuenta hasta que Countstate sea mayor que RRmin_cnt.
- Estado 5: Este estado es el estado final, en el valor de threshold va disminuyendo según el parámetro *Th.* Esta operación se lleva a cabo por medio de la instanciación de un nuevo bloque DSP48E.

$$Th = Th * \exp\ \_Th \tag{10}$$

Una vez que la muestra entrante a la máquina de estados supere el valor de este umbral se volverá al estado de Idle, comenzando de nuevo la detección del siguiente pico R.

La salida de este bloque será un pulso que marcará el pico R y un valor de cuenta que representará el tiempo entre picos o el intervalo RR.

4.2. Definición del data-path

A la hora de realizar el diseño de un DSP es de vital importancia para evitar desbordamientos, la definición de datapath o tamaño del camino de la señal en cada etapa del procesado es sin duda, el primer paso que se debe realizar a la hora de abordar un diseño como el que se trata en este trabajo.

El desbordamiento ocurre cuando un número excede el máximo valor que puede representarse con esa longitud de palabra, produciendo de esta forma un error importante puesto que el valor súbitamente pasa a 0 (aritmética sin signo), o se vuelve negativo (complemento a 2). Para evitar esta situación es preciso realizar un estudio de cada uno de los bloques de la cadena de procesado para realizar de forma correcta el dimensionamiento del datapath.

En el sistema tratado en este trabajo, el resultado de este estudio ha dado lugar a un dimensionado como el que se muestra en la siguiente figura:

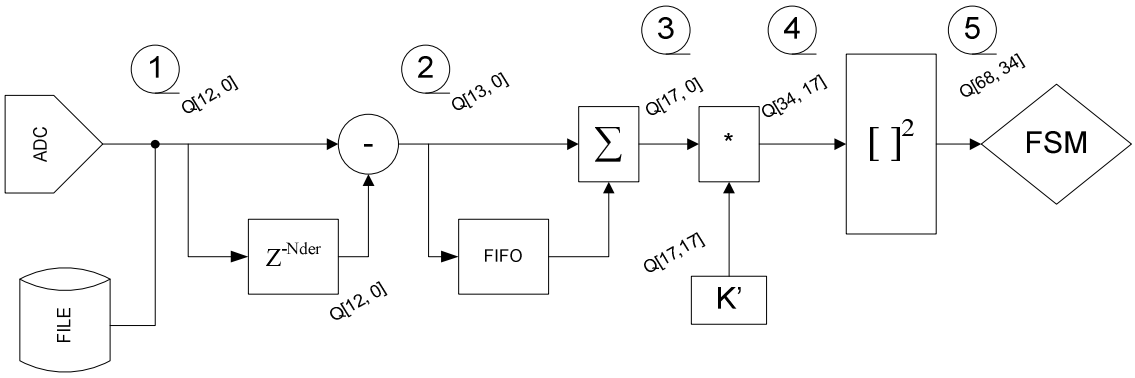

Figura 4.17 Dimensionamiento del datapath

La nomenclatura utilizada como se puede observar en cada uno de los tramos marcados es: *Q[X, Y]* donde *X* indica el ancho de palabra o número de bits totales del dato y la *Y* indica el número de esos bits que son fraccionarios. De esta forma, un dato marcado como *Q[16, 4]*, indicará que éste está expresado en un formato de coma fija de 16 bits de los cuales 4 son fraccionarios.

Para hacer más clara la explicación del porqué de la elección de un ancho de palabra u otro se ha dividido la cadena de procesado en 5 partes marcadas en la figura 4.17. A continuación se explicará cómo se ha procedido para llegar a esos valores.

- 1.- Este tamaño es el obtenido a la entrada del sistema. El dato proviene de un ADC de 12 bits. En este punto se ha decidido que la representación de este dato sea *Q[12, 0]*. De esta forma el rango dinámico que se podrá abarcar estará comprendido entre -2048 y 2047.
- 2.- Como se puede observar en este punto se efectúa una operación de resta entre dos datos *Q[12, 0].* Para evitar desbordamientos el valor de datos necesitado es de *Q[13, 0].*
- 3.- Este punto se encuentra a la salida de un acumulador. Suponiendo un máximo número de sumandos de 10 (10 sumas consecutivas) y una palabra de *Q[13, 0],* se deberá prever un número de bits igual a $\log_2 n$ bits, donde *n* en nuestro caso es 13. De esta forma nos quedaría una palabra de *Q[17, 0].*
- 4.- Este punto está a la salida de un multiplicador de dos números de 17 bits, el primero, que es el que proviene del paso anterior *Q[17, 0].* y el segundo que es *K'* que está representado como *Q[17, 17].* Al multiplicar dos números de los mismos bits se debe asegurar un resultado de 2N donde *N* en nuestro caso es 17, de esta forma el resultado estará expresado en 34 bits de los cuales 17 serán decimales *Q[34, 17].*
- 5.- El punto 5 es la salida del cuadrado. Al final lo que se hace en este tramo es multiplicar el dato del bloque anterior consigo mismo. Esto hace que el dato nuevo, siguiendo la regla de *2n* ocupará 68 bits de los cuales 34 serán decimales. O lo que es lo mismo y siguiendo con la nomenclatura empleada, se tendrá una representación *Q[68, 34].*

Este datapath fue evaluado con el modelo en coma fija realizado en matlab en capítulos anteriores con buenos resultados, por lo que queda validado para implementarlo en el sistema.

4.3. Simulaciones funcionales

En el presente punto se mostrarán las simulaciones funcionales de todo lo explicado en los puntos anteriores. Para realizar estas simulaciones se ha utilizado un testbench en el que se han instanciado el top del sistema, y un bloque cuya función es la de simular la entrega de datos del ADC.

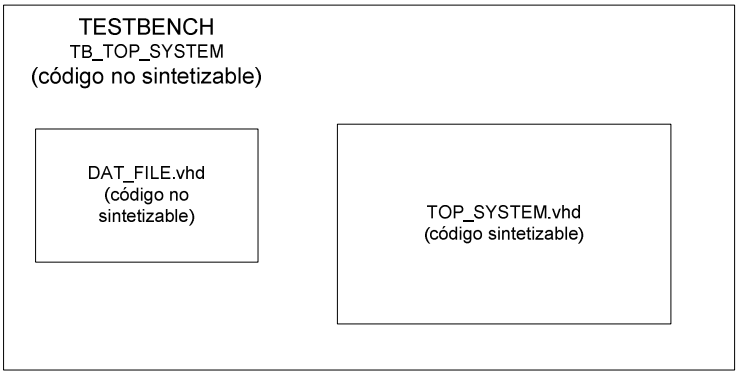

Figura 4.18 Estructura del banco de pruebas.

Esta simulación se realiza mediante la lectura de datos de un archivo en el que previamente se ha registrado una señal ECG real. Esta lectura se hace mediante el archivo de simulación DAT_FILE. El Test bench se encarga de introducir en el bloque Top_System los estímulos necesarios para el funcionamiento.

Este fichero es el mismo que se utilizó en Matlab para la realización del modelo en coma flotante.

Para la realización de este punto, se tomarán como referencia las etapas descritas en el capítulo 4.1 poniendo de esta forma resultado a lo en el descrito.

4.3.1. Entrada al sistema. Lectura del fichero

Como ya se ha indicado, los datos serán tomados de un fichero en el cual se han almacenado los datos reales de un paciente. Estas simulaciones que se van a mostrar a continuación están obtenidas a una frecuencia de muestreo de 128 Hz.

La representación analógica de la señal ECG leída de este fichero es la siguiente:

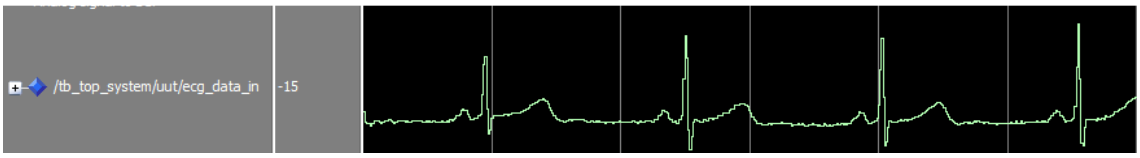

Figura 4.19 Señal recogida del fichero 16265m.dat.

4.3.2. Bloque de Configuración.

Como ya se describió en su momento, la función de este bloque es la generación de un pulso a su salida al producirse un cambio en la frecuencia de muestreo, el cual se propagará junto al dato y data valid evaluándose a la entrada de cada etapa y cambiando los parámetros según el calor elegido. Para probar esto, se ha implementado con un proceso en el test bench que cambiará este valor de forma cíclica.

```
133     stim_proc2: process
134     begin
135         Freq_sample     <=     "001";
136         wait for 1 ms;
137         Freq_sample     <=     "010";
138         wait for 1 ms;
139         Freq_sample     <=     "100";
140         wait for 1 ms;
141         Freq_sample     <=     "001";
142     end process;
```

Figura 4.20 Proceso que se ocupa del cambio del parámetro *Fs* en el TestBench

A continuación se mostrará la simulación de la generación de esta señal.

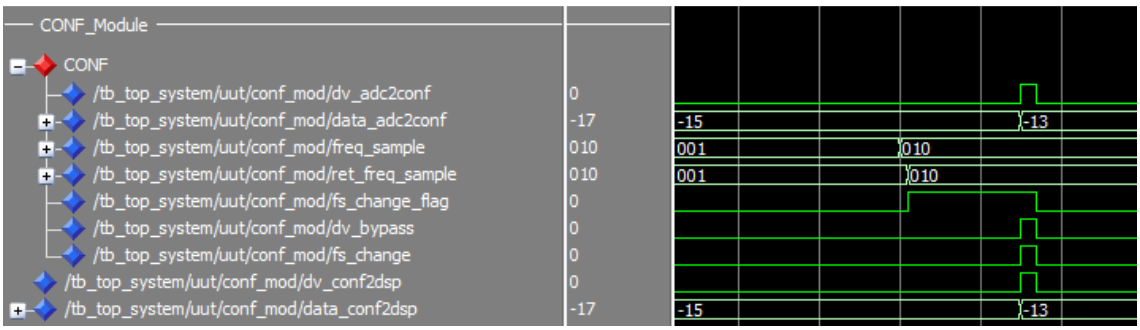

Figura 4.21 Generación de la señal fs_change.

En la figura 4.21 se aprecia cómo cambia la frecuencia de muestreo de "001", correspondiente a 128 Hz a "010" correspondiente a 256 Hz. En ese momento y gracias a un detector de cambio se levanta la señal *fs_change_flag* que se mantendrá a '1' hasta que llegue el siguiente data valid.

En ese momento la señal *fs_change_flag* se pone de nuevo a '0' esperando un nuevo cambio.

La señal de *fs_change*, se desarrolla de forma concurrente a todo esto poniéndose a '1' cuando el data valid y la señal *fs_change_flag* se encuentran a nivel alto. La tabla de verdad de *fs_change*, sería la equivalente a la de una puerta and de estas dos señales:

data valid	*fs_change_flag*	*fs_change*
1	1	1
1	0	0
0	1	0
0	0	0

Tabla 4.1. Tabla de verdad de fs_change

Una vez que el pulso en *fs_change* se ha generado, éste se propagará junto con el data valid de esa muestra a lo largo de toda la cadena de procesado evaluándose a la entrada de cada bloque.

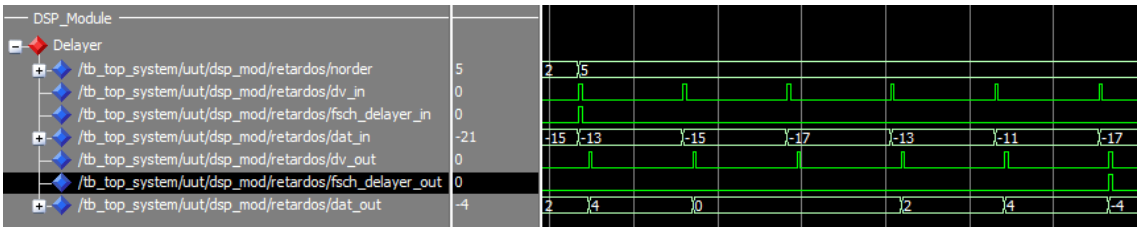

Figura 4.22 Propagación de la señal fs_change en bloque derivativo

Como se puede apreciar en la figura 4.21 con la entrada al módulo de derivación de la muestra de valor -13, el parámetro *norder* ya se ha actualizado a un valor de 5 correspondiente al de una frecuencia de muestreo de 256 Hz, esto hace que se produzca un retardo de 5 muestras.

De esta forma se puede apreciar cómo una vez pasado ese retardo, a la muestra de valor -17 se le resta la muestra con la que entró el pulso de *fs_change* al bloque obteniendo una salida de valor -4 que como se puede observar en la figura va acompañada de su data valid y el pulso de cambio de frecuencia de muestreo.

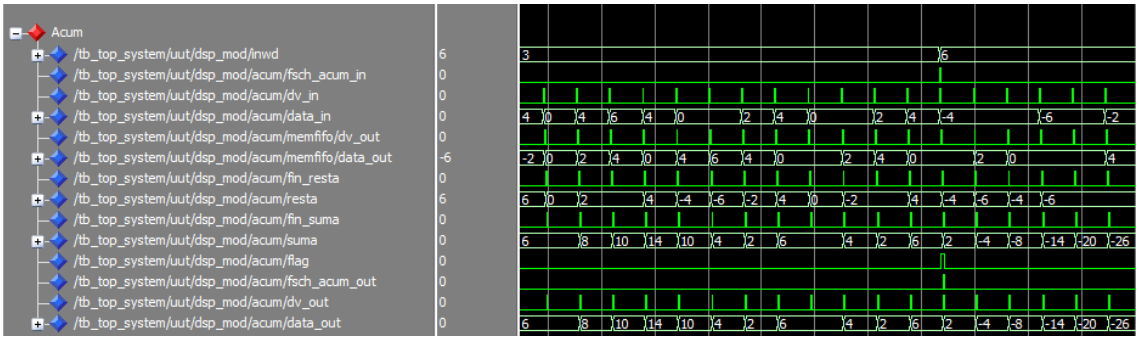

Figura 4.23 Propagación de la señal fs_change en bloque acumulador

En este caso, tal y como muestra la figura 4.23 se observa cómo el pulso de *fs_change*, aparece con el dato -4 a la entrada propagándose hacia la salida con su muestra y su data valid de nuevo.

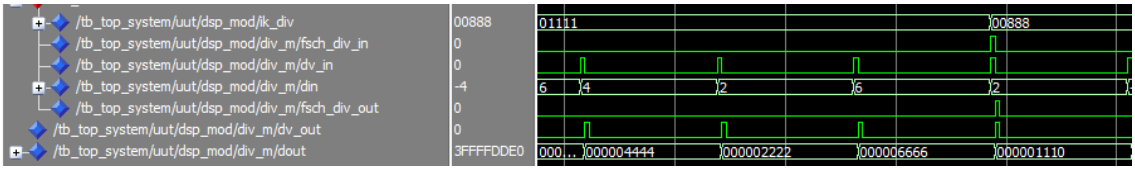

Figura 4.24 Propagación de la señal fs_change en bloque multiplicador

Al pasar al bloque multiplicador se aprecia cómo con la llegada de *fs_change* el dato *ik_div* varía y como una vez terminada la operación de multiplicación, se propaga hacia el siguiente bloque junto al data valid y la muestra.

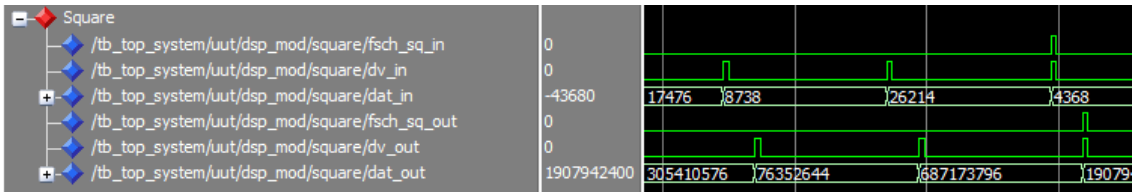

Figura 4.25 Propagación de la señal fs_change en bloque cuadrado

A continuación se muestra una figura de una señal ECG a la que se ha cambiado la frecuencia de muestreo en el transcurso de su procesado:

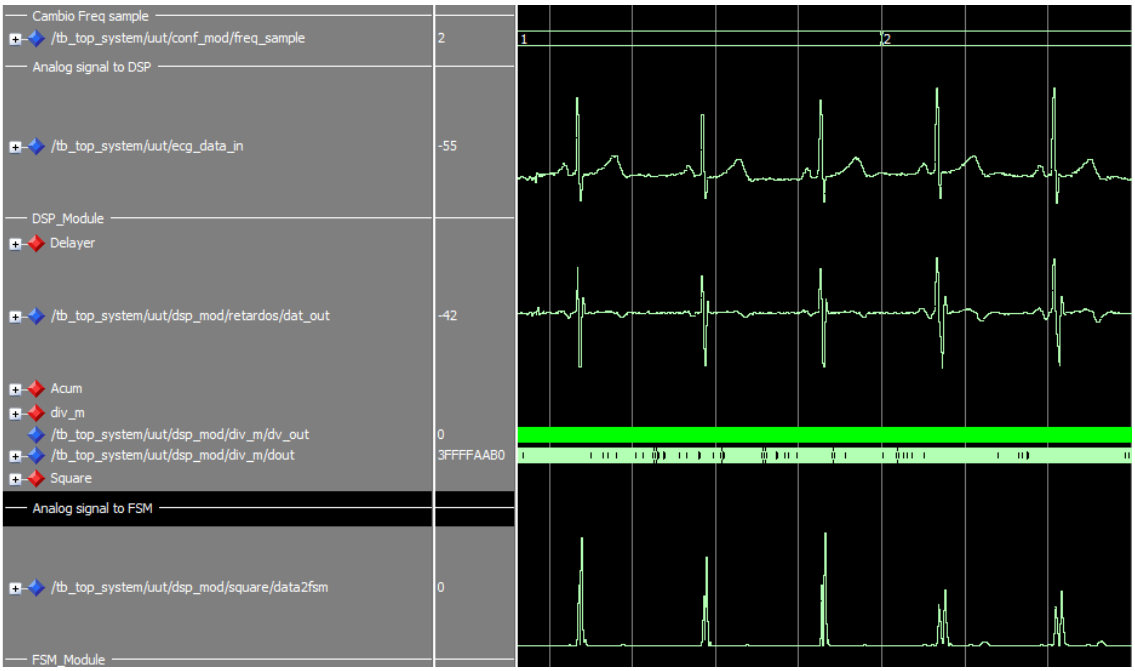

Figura 4.26 Cambio de Fs en procesado de ECG

Se puede observar cómo no se ve degradación alguna en la forma de la señal procesada. El cambio de magnitud que se aprecia en la señal de entrada a la máquina de estados se produce como consecuencia de la modificación de los parámetros del algoritmo.

4.3.3. Bloque DSP.

Como ya se comentó en la introducción a este capítulo, la señal usada para la realización de estas simulaciones, ha sido adquirida en circunstancias reales a una frecuencia de muestreo de 128 Hz. Una vez recordado este punto se pasará a simular una a una las etapas que forman el bloque DSP.

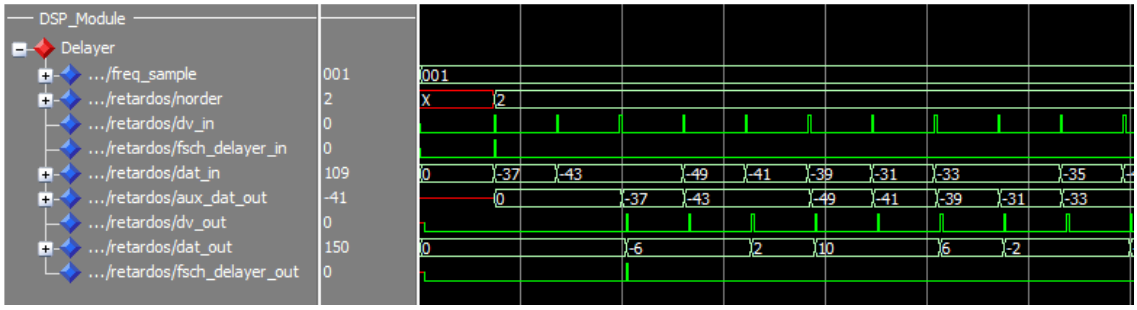

Figura 4.27 Funcionamiento fase derivativa DSP

La figura anterior muestra el funcionamiento de la etapa derivativa del bloque DSP. Se puede observar cómo con la bajada del reset a la llegada de la primera muestra válida, el sistema indica un cambio en la frecuencia de muestreo pasando (antes de que la señal llegue al bloque) el parámetro *Nder*.

Este parámetro marca el retardo que se le debe aplicar a la señal para obtener la salida, que se consigue con la resta de la señal de entrada con la señal retardada. En la figura 4.26 se puede ver este proceso donde la señal aux_dat_out corresponde a la señal retardada y norder al parámetro *Nder*. El resto de señales vienen identificadas por su nombre

Una vez aplicada esta etapa la señal resultante sería la mostrada a continuación:

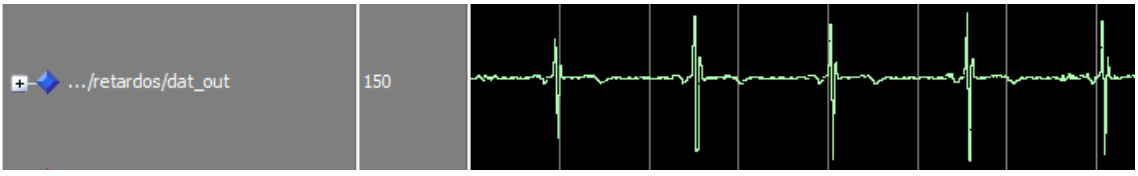

Figura 4.28 Salida analógica del bloque derivativo.

Esta etapa, como se puede observar, acentúa las pendientes tan características del complejo QRS, lugar donde se encuentra el pico R. A continuación la señal se introduce en la fase donde se desarrolla la ventana de integración (Acumulador).

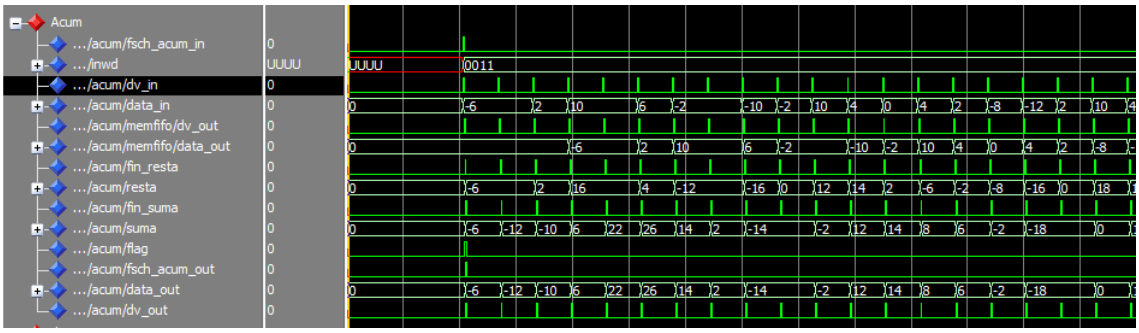

Figura 4.29 Funcionamiento fase acumulativa.

En esta etapa lo que se pretende es crear un acumulador dinámico, que acumule un número de datos fijo en el tiempo. Esto se ha conseguido hacer creando una FIFO de n elementos. Los datos van entrando en esta FIFO y cuando se llena empiezan a salir. A su salida, se resta el saliente con el entrante y el resultado se suma a la acumulación de esos N elementos. De esta forma:

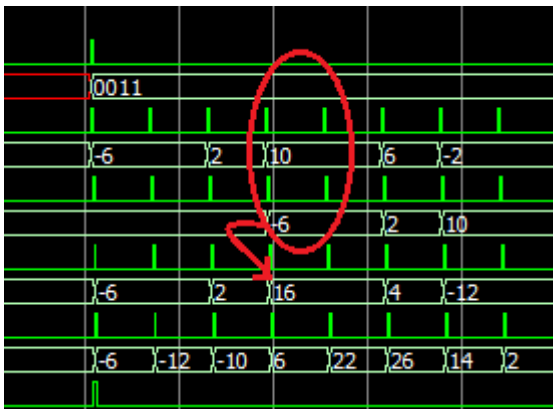

Figura 4.30 Resta dato entrante con saliente.

En la figura 4.30 se observa cómo se produce la resta entre el dato entrante 10 y el saliente -6 dando como resultado 16.

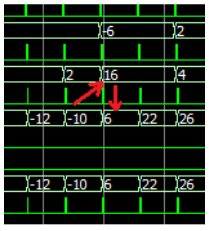

Figura 4.31 Suma de la resta con la acumulación de 3 muestras.

Finalmente este resultado se suma al dato resultante de la acumulación de 3 muestras, en este caso -10, y el resultado ya es el dato que va a la fase siguiente.

La siguiente etapa consiste en una división. Como ya se dijo en su momento las divisiones tienen un coste de recursos elevados en los dispositivos programables, es por esto que se optó por multiplicar por la inversa del divisor.

De esta forma se obtuvo el parámetro *iK_div* y se multiplicó por el dato proveniente de la fase anterior.

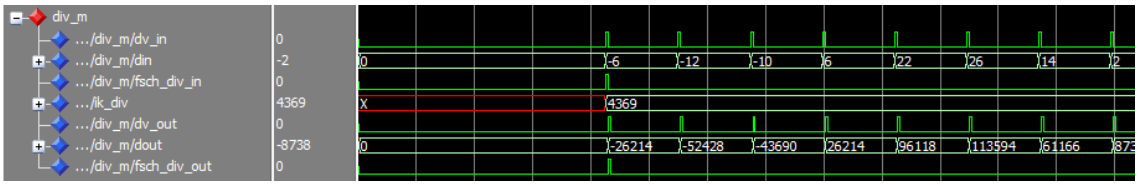

Figura 4.32 Multiplicación dato por inversa del parámetro K.

El parámetro *iK_div* se corresponde en este caso al número decimal 0.03333. Al estar representado en punto fijo con 17 bits (todos ellos decimales) el dato de *iK_div* mostrado en el simulador debe ser dividido entre 2^{17} para conocer su magnitud real.

Una vez realizada esta multiplicación y reconstruyendo la señal a su formato analógico se obtiene el siguiente resultado (la señal ha sido truncada para su representación):

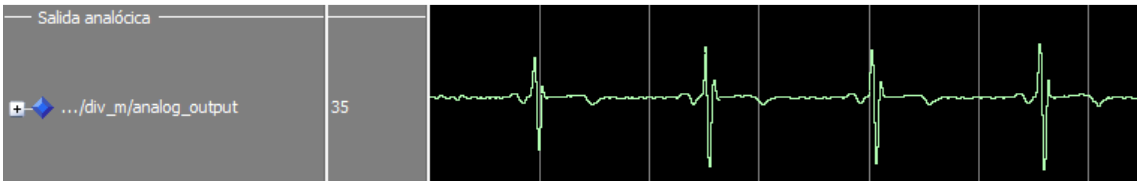

Figura 4.33 Salida analógica del bloque acumulativo + multiplicativo.

Lo que se puede observar en esta figura es la eliminación de la mayor parte de las oscilaciones y picos que no corresponden al complejo QRS. Después de todo esto, cada muestra de la señal es elevada al cuadrado en el siguiente bloque quedando la señal de la siguiente forma:

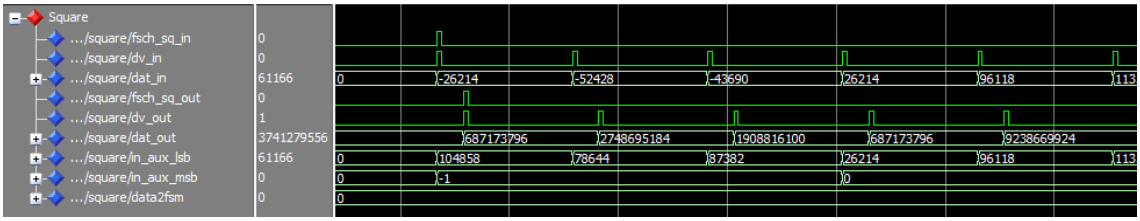

Figura 4.34 Elevación al cuadrado de la señal.

El dato en este punto es de 68 bits de los cuales 34 son enteros. Para poder manejar estos datos con mayor facilidad, la salida de este bloque se trunca en 32 bits. Esta señal con los datos ya truncados es la llamada data2fsm.

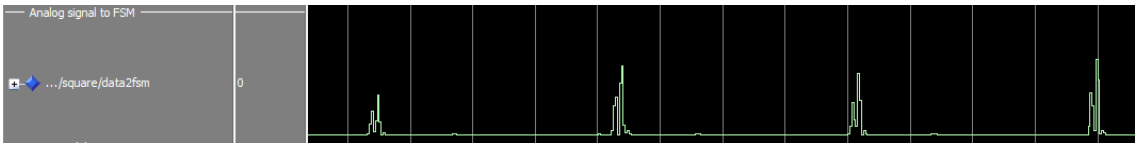

Figura 4.35 Entrada a la máquina de estados.

Al elevar al cuadrado la señal conseguimos que esta sea totalmente positiva y acentuar mas los picos que queremos detectar. Una vez llegados a este punto, la etapa de pre-procesamiento llega a su fin y los datos se entregan a la máquina de estados para la detección de los picos.

4.3.4. Bloque FSM.

Este bloque es el que implementa el algoritmo de detección.

Se trabajará con los tiempos de RR_{min} y QRS_{int}. El valor de estos tiempos es de 200 ms y 60 ms respectivamente. Para acortar tiempos de simulación se redujeron junto a la frecuencia de muestreo de esta forma en un factor de 1/12500. quedando de esta forma unos tiempos de 16 µs para RR_{min} y 4.8 µs para QRS_{int}.

Es importante apuntar que aunque para la simulación se acortaron los tiempos y de esta forma la cuenta de los contadores implementados para cumplirlos. Estos contadores fueron dimensionados para poder respetar los tiempos reales. De esta forma con solo cambiar los valores de cuenta que se

encuentran en el código del archivo FSM_module.vhd respetaría los tiempos marcados.

```
RRmin_cnt         <= x"000320" ;   -- 16us   - cuenta real (0xF42400)  factor: 1/12500
QRSint_cnt        <= x"0000F0" ;   -- 4.8us  - cuenta real (0x44AA20)  factor: 1/12500
RRmin_QRSint_cnt  <= x"000410" ;   -- 20.8us - cuenta real (0x13D6200) factor: 1/12500
```

Figura 4.36 Asignaciones de valores de cuenta.

Una vez que esto ha quedado claro, se puede pasar a ver los resultados de las simulaciones para este bloque.

Al empezar el sistema, la máquina de estados permanece en un estado de Idle hasta que le llega el primer data valid, en ese momento la señal dv_dsp2fsm_detector se pone a nivel alto haciendo que a partir de entonces el estado Idle se comporte como un estado de transición. (se debe recordar que esta señal es la condición para el cambio de estado). También merece la pena recordar que es en este estado donde se evalúa el cambio de frecuencia y en caso de ser necesario se actualiza la señal *exp_Th.*

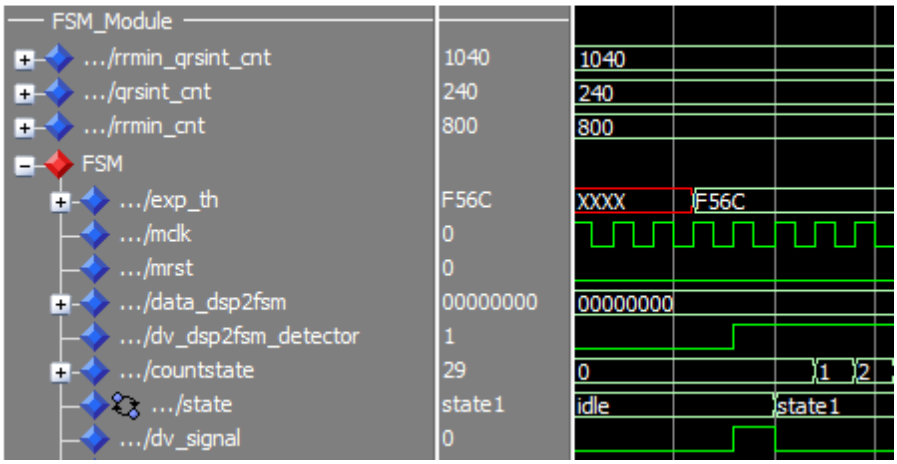

Figura 4.37 Estado Idle de máquina de estados.

Una vez se pasa al estado 1, el contador *countstate* comienza una cuenta hasta llegar al número de ciclos requeridos para cumplir el tiempo QRS_{int} + RR_{min}. Durante este periodo se van evaluando los datos en la búsqueda de máximos locales.

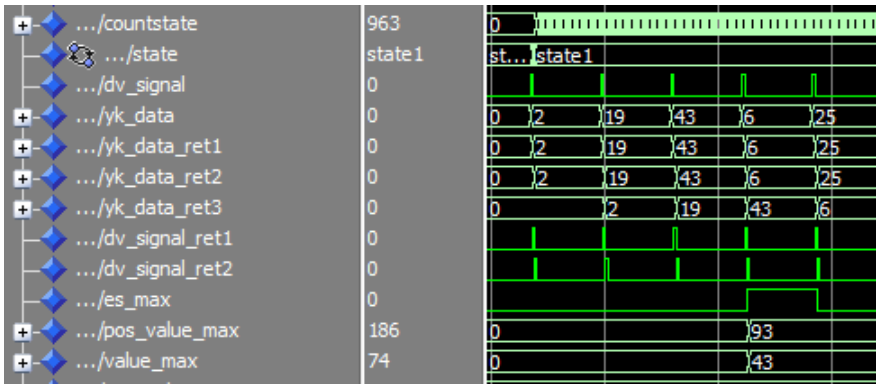

Figura 4.38 Levantamiento de es_max.

Esta detección se realiza de la siguiente forma, se retarda la señal una muestra (*yk_data_ret2 e yk_data_ret3*) y se realiza una comparación. Cuando la señal retardada es mayor que la no retardada, en nuestro caso *yk_data_ret2* > *yk_data_ret3*, entonces eso significa que existe un máximo local, en ese momento, la señal *es_max* pasa a nivel alto hasta que esta situación se revierte. Sin embargo el valor y la posición que provocó esta situación se almacena en *pos_value_max* y en *value_max* y permanecerán almacenados hasta que no se encuentre otro máximo local mayor que el encontrado.

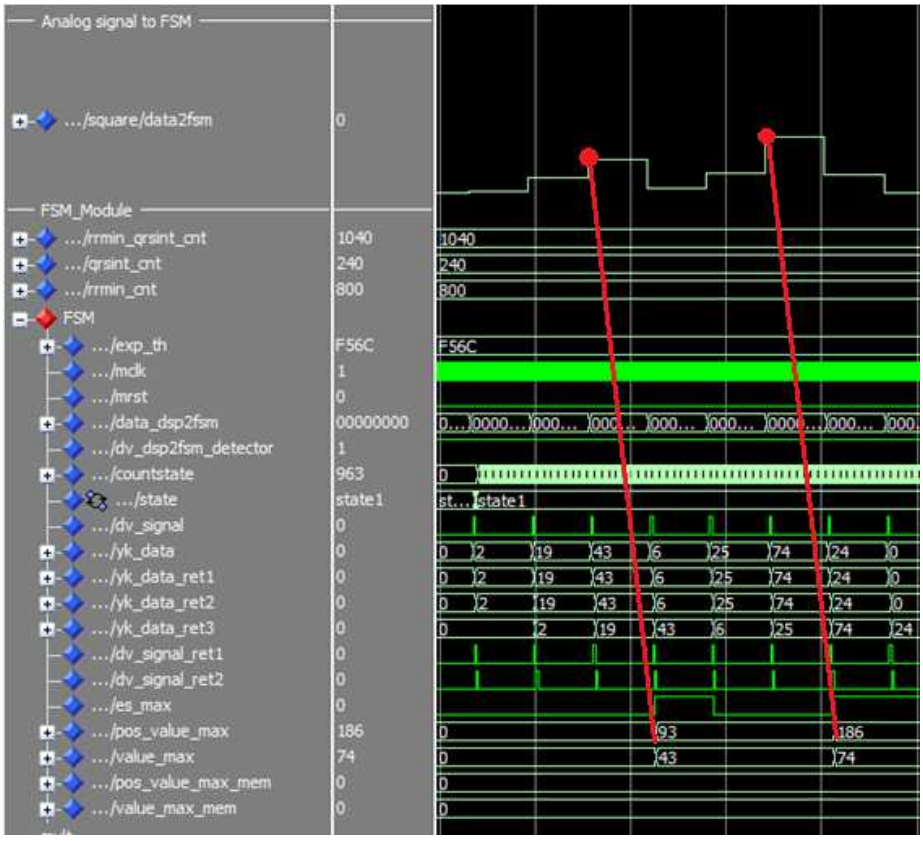

Figura 4.39 Actualización de pos_value_max y value_max.

De esta forma al final de este estado se tendrán almacenadas en estas señales el máximo valor junto a su posición encontrados en el estado 1.

Una vez se llega al estado 2, este máximo se almacena en *pos_value_max_mem* y *en value_max_mem*. Al mismo tiempo que se realiza la operación:

$$countstate = countstate - (pos\_value\_\max + 1) \quad (9)$$

Con esto se deja preparado el valor de arranque de *countstate* para la cuenta del estado 4. Este estado es un estado de transición y dura 1 ciclo de reloj.

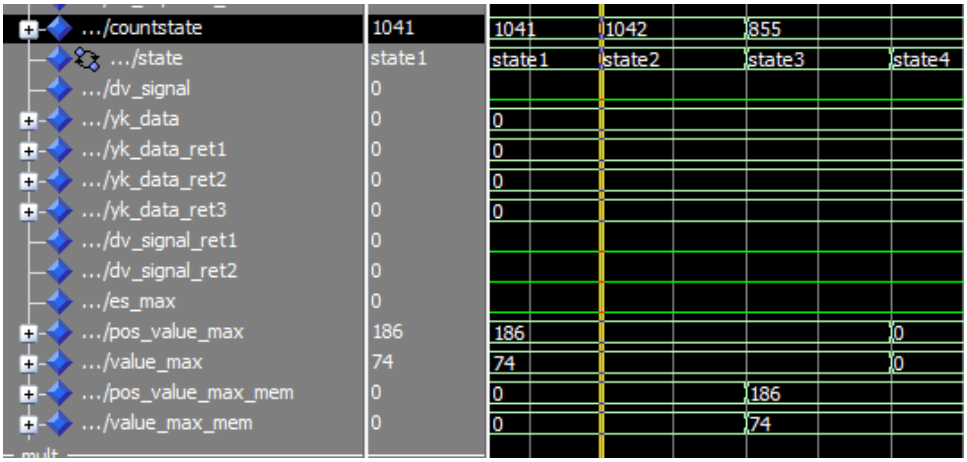

Figura 4.40 Fijación de inicio de cuenta y almacenamiento de pico máximo en pos_value_max_mem y value_max_mem.

Realizadas estas acciones y tras un ciclo de reloj, se pasa al estado 3. En este estado se realiza el cálculo del parámetro *Th*

$$Th\_calc = \frac{Th\_signal + value\_\max\ \_mem}{2} \quad (10)$$

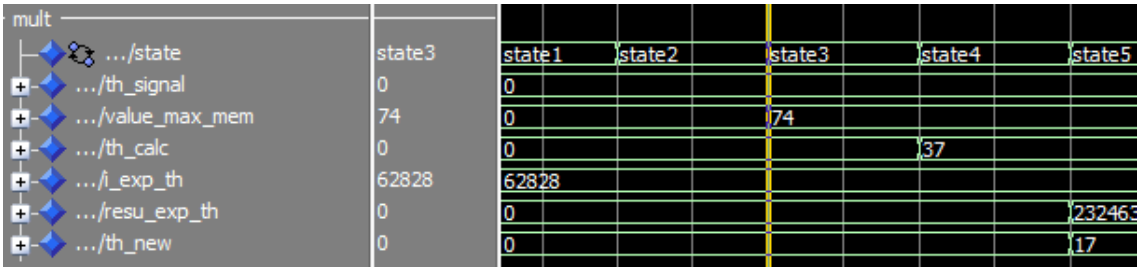

Figura 4.41 Cálculo de Th_calc.

En el estado 4 se realiza una cuenta con el valor precargado en el estado 2 hasta llegar a RR_{min}. Una vez superado este estado, se pasa al estado 5.

Al finalizar la cuenta del estado 4 se producirá un pulso que indicará la presencia de un pico R. Esta señal estará retardada con respecto a su fuente de origen, pero debido a que estará retardada el mismo tiempo en todos los picos encontrados, la distancia entre ellos será la misma que entre los picos R sin retardar, por lo que nos permitirán saber el tiempo que hay entre ellos y de esta forma la frecuencia cardiaca del paciente, la cual era uno de los objetivos buscados en este trabajo.

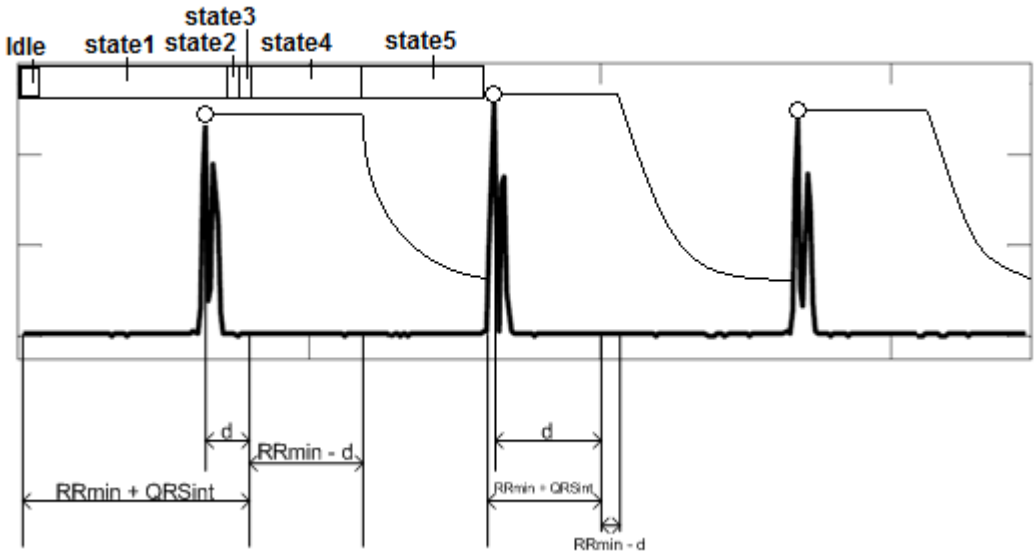

Figura 4.42 Progresión de los estados de la FSM.

Mostrando lo anteriormente expuesto de forma gráfica, se puede observar que el pico siempre se encuentra a una distancia RR_{min} del estado 5 por lo que poniendo el pulso que indica pico R en el último ciclo de reloj del estado 4, se consigue que la distancia entre pulsos represente fielmente la distancia entre picos detectados.

Y por último, en este estado se fija el valor de threshold a través de la siguiente fórmula:

$$Th\_new = i\_exp\_Th * Th\_calc \quad (11)$$

Una vez que el valor del dato a la entrada de la máquina de estados sobrepase el valor de *Th_new*, la máquina de estados pasará a Idle y se iniciará un nuevo ciclo de detección.

La salida del sistema serán los pulsos generados a partir de los picos R y un registro de 32 bits que muestra la cuenta de ciclos de reloj que hay entre picos. De esta forma se puede obtener el intervalo RR.

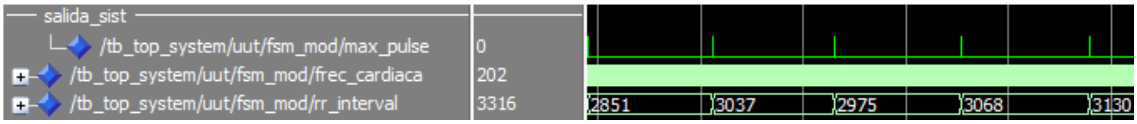

Figura 4.43 Salida del sistema.

Capítulo 5. Resultados

En este capítulo se mostrarán los resultados del trabajo obtenidos. Estos resultados serán comprobados por dos vías:

1. La primera consiste en las simulaciones temporales del algoritmo ya implementado sobre la FPGA, estas simulaciones emulan la arquitectura interna del dispositivo de destino por lo que un correcto funcionamiento en este punto, asegura el correcto funcionamiento en el dispositivo FPGA final.
2. La segunda, validará la etapa de preprocesado. Se generarán en cada etapa unos archivos, que registrarán la forma de la señal en ese punto y con estos se creará un script en Matlab que los compare con los obtenidos en el modelo de coma fija. El resultado en ambos archivos debe ser el mismo, si esto ocurre, se podrá afirmar que esa partede la cadena es correcta.

Dicho esto, a continuación se procederá a mostrar los resultados obtenidos en cada uno de los puntos.

5.1. Simulaciones temporales

Una vez realizada la implementación del diseño:

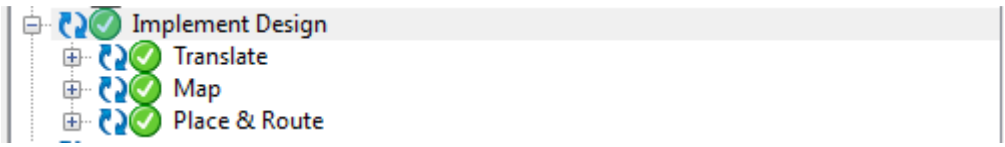

Figura 5.1 Ventana de proceso del IDE.

Se procederá a lanzar el Modelsim en el modo Post-Route:

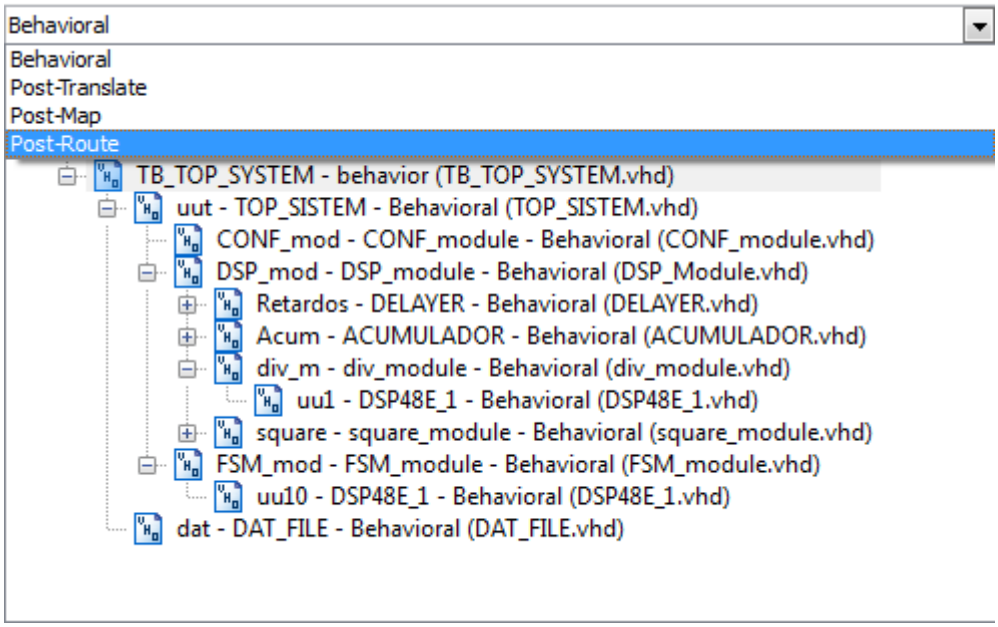

Figura 5.2 Elección de modo post-route para simulación.

Acto seguido se lanzará la simulación.

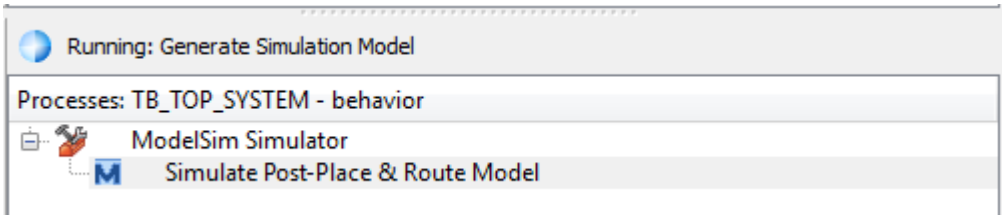

Figura 5.3 Lanzamiento de Modelsim en modo post-route.

Realizar esta simulación es importante ya que aunque la funcional haya mostrado un correcto funcionamiento del sistema, es posible que al implementarlo no funcione correctamente. Una de las posibles causas de esto son los retardos internos del dispositivo. Esta simulación emula la arquitectura interna del chip, teniendo en cuenta estos retardos y permitiendo conoce si todo está correcto o por el contrario existen errores que hagan retroceder a pasos anteriores para solventarlo.

La simulación temporal del sistema es la siguiente:

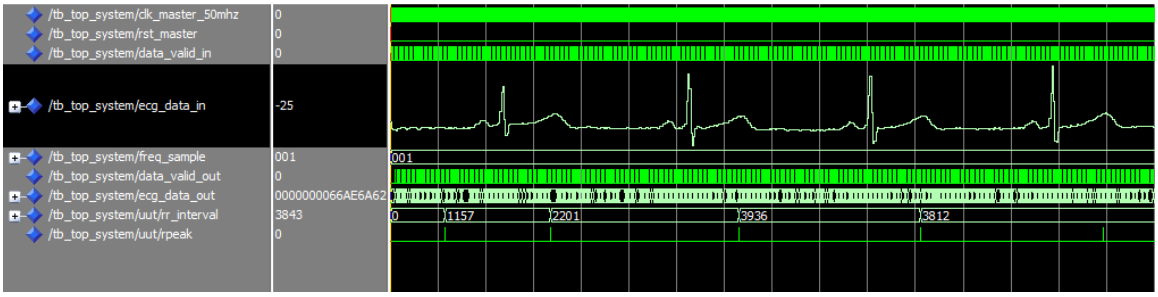

Figura 5.4 Simulación temporal del sistema.

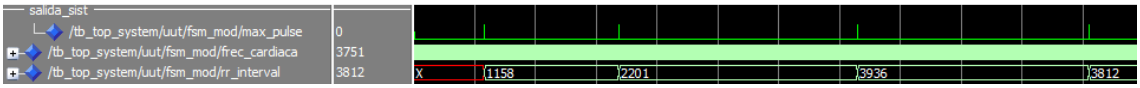

Figura 5.5 Salida de la simulación funcional del sistema.

Como se puede observar la simulación devuelve la cuenta correspondiente al intervalo RR y el pulso indicador de pico R. La señal de entrada como se puede apreciar está adelantada con respecto a la salida. Esto se debe a que debido a toda la latencia introducida por la etapa de procesado la salida sale con un cierto retardo. En la figura 5.5. se muestra de nuevo los resultados obtenidos para la simulación funcional, habiendo obtenido en ambas los mismos resultados

La etapa de procesado se comprobará en el punto siguiente. Para hacer esto se ha insertado código no sintetizable a la salida de cada bloque. La función de este código es la de generar un archivo con el valor de todas las muestras de la señal. De esta forma se podrá introducir estos datos en Matlab y comprobar de esta forma si la implementación se realizó correctamente.

Como he dicho el código empleado en el diseño de estos bloques no es sintetizable. Para evitar problemas a la hora de implementar, se ha utilizado la sentencia pragma translate off / pragma translate on. Esto permite indicar a la herramienta que esas líneas no deben ser tenidas en cuenta por el sintetizador.

```
--pragma translate_off
  process
  -- Tratamiento del fichero de salida
  file InputFile1 : datos_salida open write_mode is "vhd_delayer.dat";
  variable dato_escrito: integer;

  begin

  for i in 0 to 20000 loop     -- rango de muestras tomadas
     wait until mclk = '1' and mclk'event;
        if ret2_iDV_OUT = '1' then
           dato_escrito := to_integer(signed(iDAT_OUT));
           write (InputFile1, dato_escrito);
        end if;
  end loop;

  file_close(InputFile1);
  wait;
  end process;

--pragma translate_on
```

Figura 5.6 Ejemplo de uso de la sentencia pragma translate off/on.

La figura 5.6. hace referencia a la salida del bloque derivativo. Como se puede ver se usan sentencias no sintetizables como el bucle for, sin embargo gracias a la sentencia pragma translate esto no será ningún problema a la hora de dejar estos procesos en el diseño.

5.2. Matlab vs Modelsim. Comprobación del bloque DSP

Como ya se comentó en el punto anterior, el preprocesado de la señal se comprobará con un script realizado en Matlab. Este script tomara los archivos generados durante la simulación temporal y los comparará con los obtenidos en el modelo del algoritmo en coma fija que se modeló al principio de este trabajo.

De esta forma se va a realizar la comprobación de cuatro etapas. La primera se hará después de la etapa derivativa, la segunda se realizará después del acumulador, la tercera será la multiplicación por el parámetro K' y última será ya la salida del DSP y la entrada a la máquina de estados.

5.2.1. Fase derivativa.

En esta fase se obtuvieron los siguientes resultados:

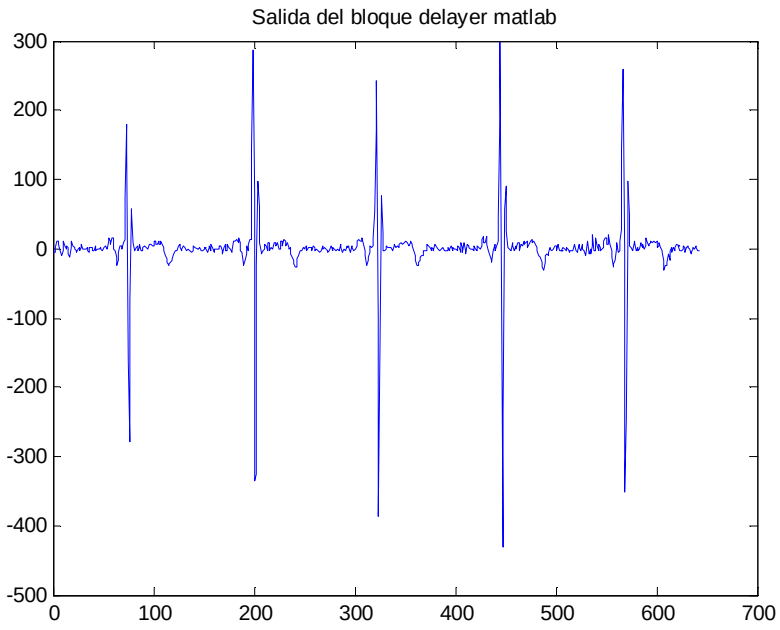

Figura 5.7 Salida del bloque derivativo de la señal de Matlab.

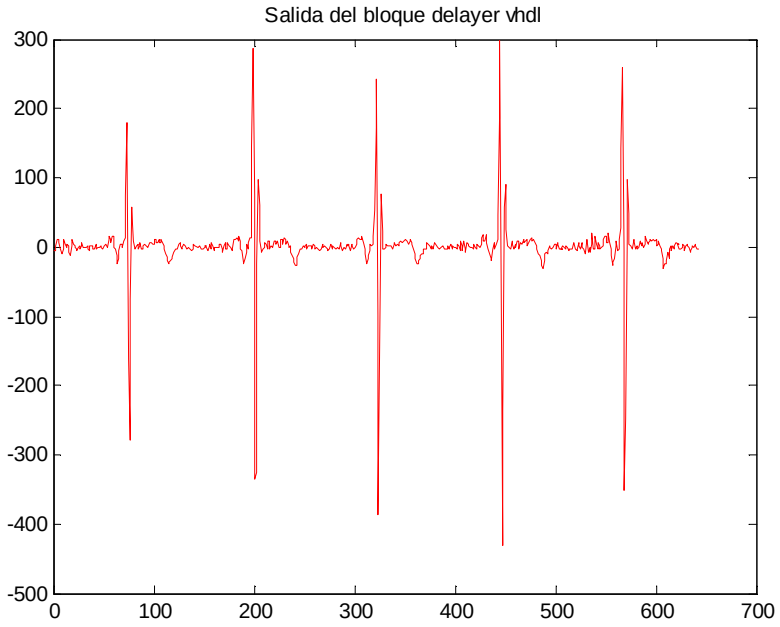

Figura 5.8 Salida del bloque derivativo de la señal de FPGA.

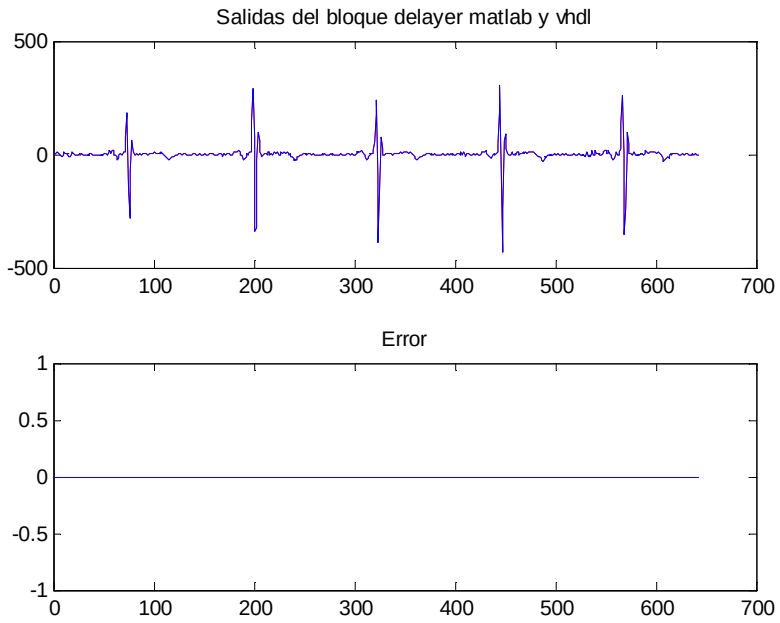

Figura 5.9 Comparación de las señales de salida del bloque derivativo de Matlab y de la FPGA.

Se puede observar como ambas señales son iguales, la figura 5.7 corrobora esta afirmación, mostrando en la primera imagen ambas señales superpuestas, y en la segunda la diferencia entre ellas.

Hay que comentar que para que ambas señales cuadraran hubo que retrasar la obtenida en Matlab un total de tres muestras. Esto se debe a que Matlab al trabajar con ficheros, puede disponer de los datos instantáneamente, por lo que no sufre retardos, pero al trabajar sobre una plataforma real, las tres muestras de latencia del inicio de este bloque necesarias para empezar a sacar dato válido (recordemos que la salida de este bloque era la diferencia entre la muestra Y_k y la muestra $Y_{k\ -3}$). Este retardo de tres muestras se vendrá propagando por toda la cadena.

5.2.2. Fase acumulativa.

En esta fase se obtuvieron los siguientes resultados:

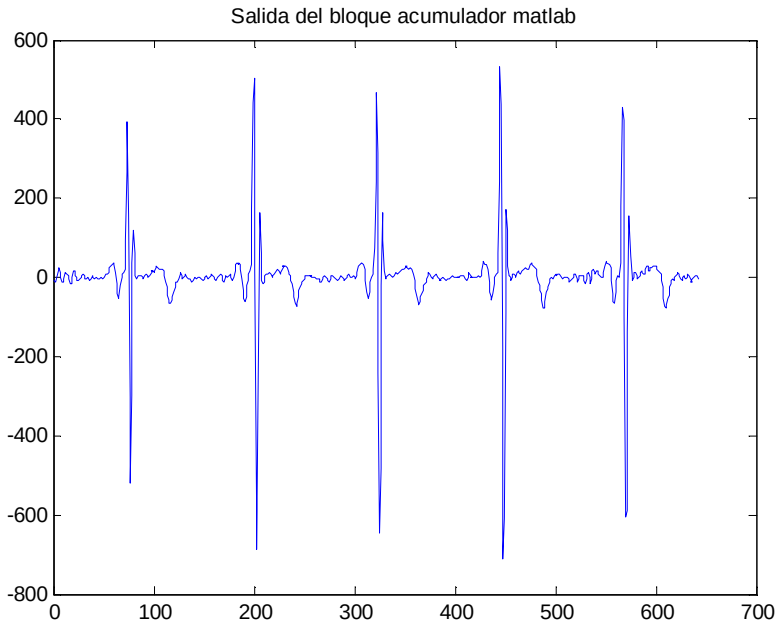

Figura 5.10 Salida del bloque acumulador de la señal de Matlab.

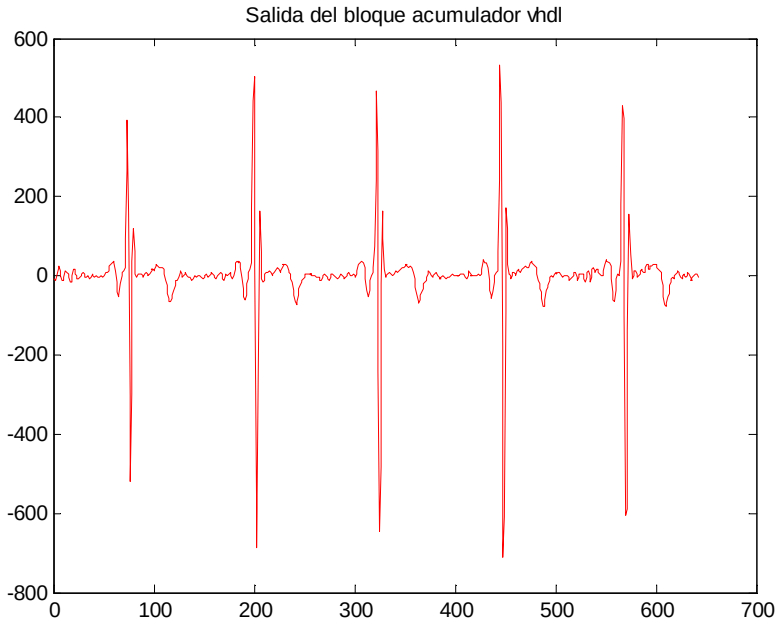

Figura 5.11 Salida del bloque acumulador de la señal de FPGA.

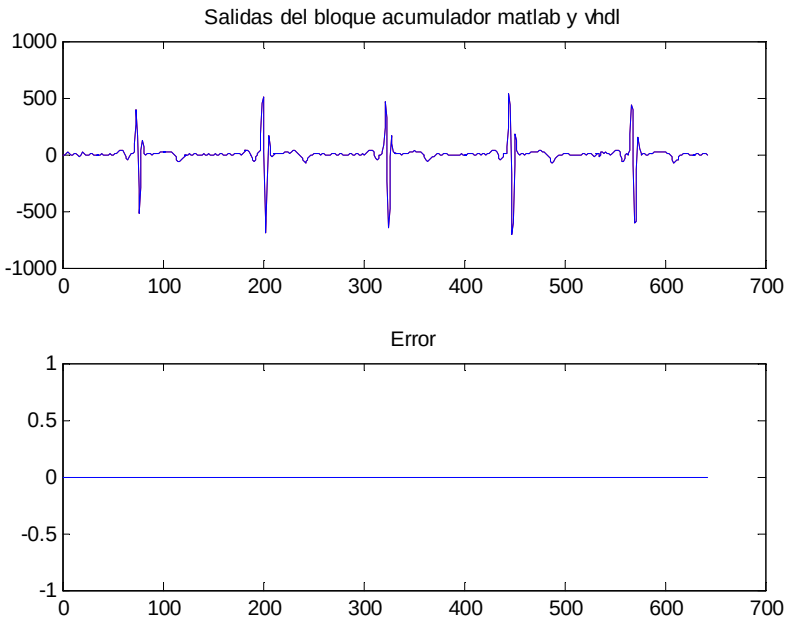

Figura 5.12 Comparación de las señales de salida del bloque acumulador de Matlab y de la FPGA.

Al igual que se hizo con la fase derivativa, se obtenido las señales de ambas fuentes (Matlab y FPGA) y se han comparado superponiéndolas y obteniendo la diferencia entre ambas. El resultado es un error igual a 0. Esto verifica que esta fase está correctamente implementada.

5.2.3. Multiplicación por k'.

En esta fase se obtuvieron los siguientes resultados:

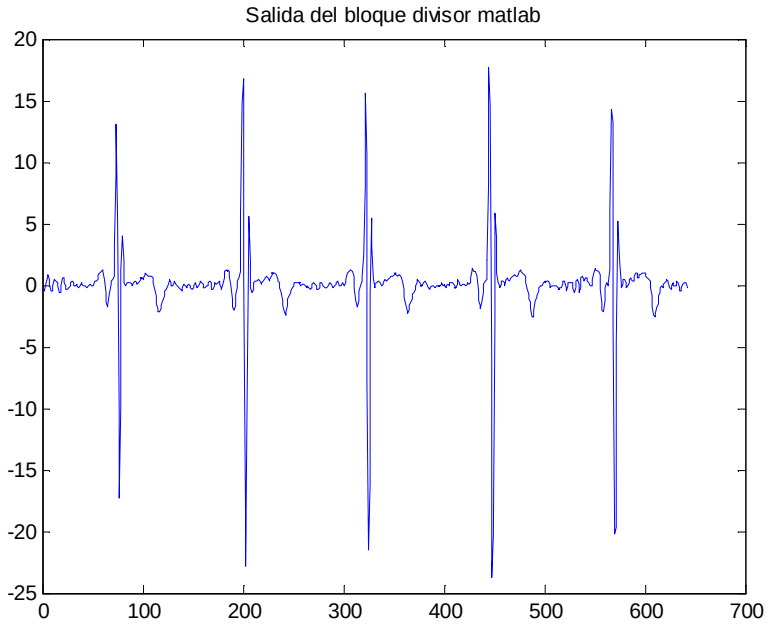

Figura 5.13 Salida del bloque multiplicador de la señal de Matlab.

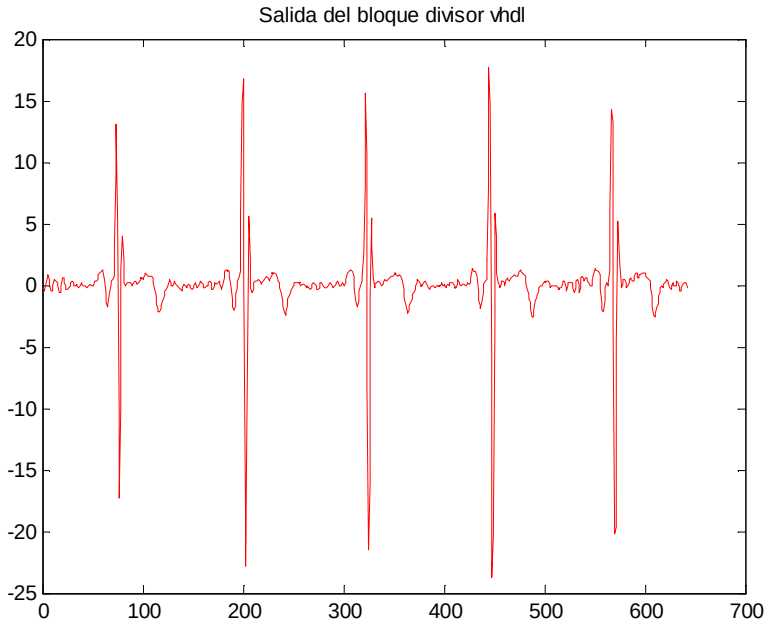

Figura 5.14 Salida del bloque multiplicador de la señal de FPGA.

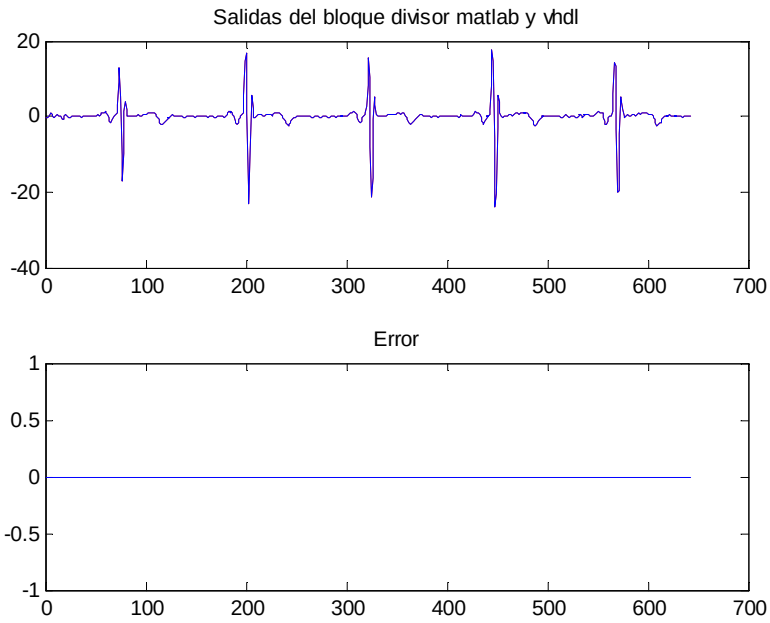

Figura 5.15 Comparación de las señales de salida del bloque multiplicador de Matlab y de la FPGA.

Aunque las imágenes tengan como título bloque divisor, en esta etapa lo que se realiza es una división de la inversa del parámetro K al que se ha llamado K'. La medida de la figura 5.14 se ha tenido que reescalar, ya que al multiplicar dos número de 17 bits, uno con todos los bits enteros y el otro con todos los bits decimales el resultado era un número de 34 bits de la forma Q[34, 17] (34 bits de los cuales 17 son decimales.). Por esta razón este dato tuvo que ser dividido por 2^{17} lo que como es de sobra conocido, equivale a un desplazamiento de 17 bits hacia la derecha del dividendo.

Una vez hecho esto y como se puede apreciar en la figura, el error obtenido es 0 y las dos señales coinciden perfectamente.

5.2.4. Cuadrado.

En esta fase se obtuvieron los siguientes resultados:

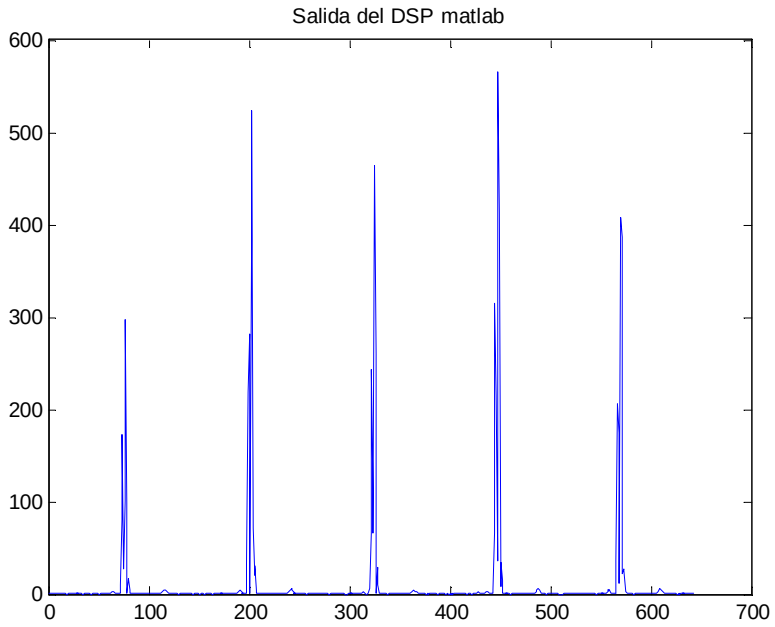

Figura 5.16 Salida del bloque cuadrado de la señal de Matlab.

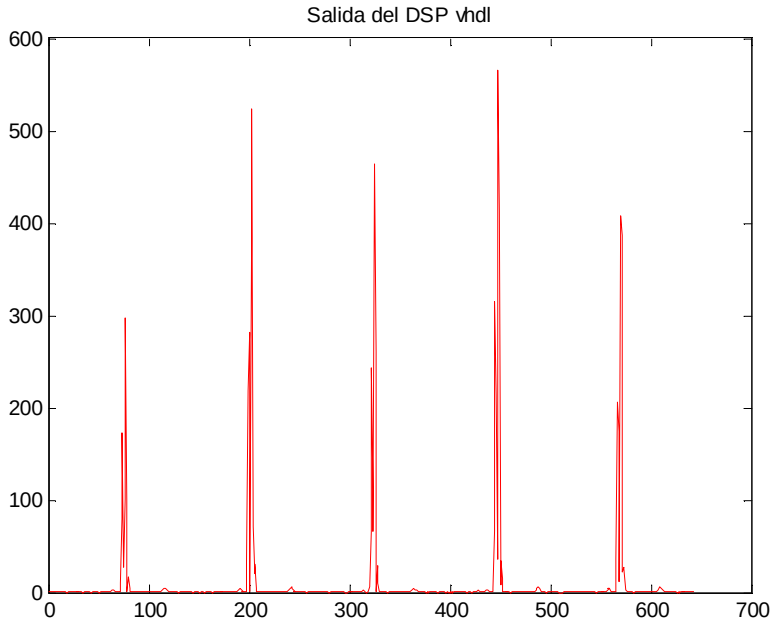

Figura 5.17 Salida del bloque cuadrado de la señal de FPGA.

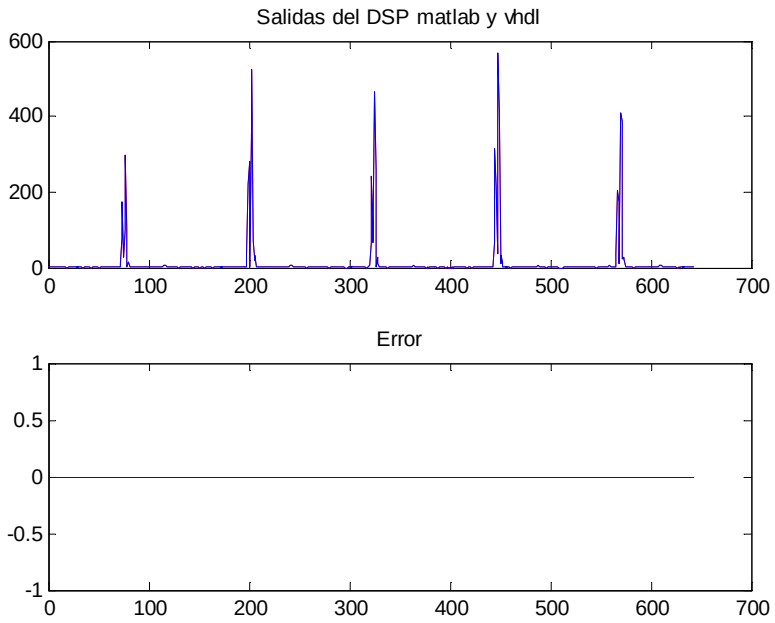

Figura 5.18 Comparación de las señales de salida del bloque cuadrado de Matlab y de la FPGA.

En este caso pasa lo mismo que en el paso anterior, debido a que la salida de este bloque está representado por un número de 68 bits de los cuales 34 son decimales, para que coincidieran las escalas fue necesario dividir el dato recogido de la FPGA por 2^{34} de esta forma al compararlos tal y como se muestra en la figura 5.18. se obtiene que las señales son iguales y el error es 0.

Con la verificación de esta etapa, queda validado el bloque del DSP ya que se ha conseguido lo que se pretendía, una implementación exacta del modelo construido en Matlab en el formato de coma fija.

5.2.5. Otros resultados

Aparte de los ya comentados existen otros resultados que merece la pena mencionar, como por ejemplo el resultado de hacer el dato de entrada a la máquina de estados más manejable. Como ya se dijo en su momento el dato saliente del DSP tiene un tamaño de 68 bits. El manejo de este tamaño de palabra en el bloque de detección era bastante complicado, por lo que se decidió ver el efecto que podía tener truncarlo para quedarse con un tamaño de palabra de 16 bits. Tras hacer las simulaciones en Matlab se llegó a los siguientes resultados:

```
*** Paciente 16272 coma flotante***
  Total peaks: 7988
      TP: 7985
      FN: 3
      FP: 9
        * Se: 99.962444
        * P+: 99.887416
```

Figura 5.19 Resultados modelo coma flotante.

```
*** Paciente 16272 coma fija***
  Total peaks: 7988
      TP: 7985
      FN: 3
      FP: 15
        * Se: 99.962444
        * P+: 99.812500
```

Figura 5.20 Resultados modelo coma fija.

Como se pudo observar este cambio apenas afectaba a los resultados, se produjo un mínimo aumento de los falsos positivos y empeoró un poco la predectividad positiva, por lo que después de valorarlo, se decidió truncar el dato.

Otro aspecto a tener en cuenta son los recursos de la FPGA empleados para el desarrollo del algoritmo. Para ello una vez implementado el diseño, se pueden ver los datos ofrecidos por el ISE, la plataforma de desarrollo de Xilinx, para ver cómo se han distribuido los recursos de la pastilla.

De esta forma viendo los reports detallados del Place and Route se puede apreciar los siguientes resultados:

```
Device Utilization Summary:

   Number of BUFGs                           1 out of 32      3%
   Number of DSP48Es                         6 out of 384     1%
   Number of External IOBs                 120 out of 960    12%
      Number of LOCed IOBs                   0 out of 120     0%

   Number of Slices                        314 out of 30720   1%
   Number of Slice Registers               856 out of 122880  1%
      Number used as Flip Flops            856
      Number used as Latches                 0
      Number used as LatchThrus              0

   Number of Slice LUTS                    443 out of 122880  1%
   Number of Slice LUT-Flip Flop pairs    1022 out of 122880  1%
```

Figura 5.21 Recursos utilizados.

Capítulo 6. Conclusiones y trabajos futuros

Y aquí llega a su fin este trabajo. De forma general, se han cubierto los objetivos marcados al principio del mismo, es decir, se ha conseguido implementar un algoritmo de detección de picos R sobre un dispositivo FPGA de la familia Virtex5 de Xilinx de gran robustez y que utiliza un número bajo de recursos para llevar a cabo esta detección en tiempo real en dispositivos básicos que no dispongan de una potencia de procesado elevada.

En cuanto a lo que se refiere a los objetivos que subyacen por debajo del propósito general comentado anteriormente, se puede afirmar que:

- Se ha definido y caracterizado de forma bastante detallada la señal biomédica a procesar, resaltando sus características más interesantes para llevar a cabo la detección de los picos R.
- Se ha definido el datapath empleado, confirmando a través del modelo en Matlab en coma fija que no existiera una pérdida de resolución importante que pudiese corromper los resultados finales.
- Se ha implementado un bloque de procesado robusto y eficiente, así como un bloque detector con las salidas necesarias para obtener mediante el método elegido la frecuencia cardiaca del paciente o intervalo RR.

También en este punto, merece la pena comparar el algoritmo empleado con el algoritmo de Pan & Tompkins, de esta forma se puede afirmar que la etapa de preprocesado en el sistema propuesto en este trabajo es mas liviano en cuanto a lo que el uso de recursos se refiere, obteniendo unos resultados válidos para el objetivo propuesto.

Si se habla sobre futuros trabajos en lo concerniente a este trabajo, éstos están claros, el algoritmo ya ha sigo implementado por lo que lo que hace falta es la monitorización de las salidas del mismo.

En este trabajo se han dejado preparadas dos salidas del sistema: un pulso que indica detección de pico R y un valor de cuenta que indica el intervalo RR para este propósito. Una idea que es tendencia actualmente, podría ser aprovechar la potencia de cálculo de los microprocesadores y crear un System on Chip con esta parte de procesado y detección ya implementada. Podría ser de interés, aprovechando el bajo número de recursos que usa este diseño, implementar en la misma pastilla un softprocesor como puede ser por ejemplo el Microblaze de Xilinx y pasarle el tiempo del intervalo RR para que éste fuera el que realizara la conversión del valor de cuenta a tiempo.

Igualmente se podría desarrollar una aplicación html y monitorizar en ella el resultado. Este resultado podría ser visible desde cualquier ordenador con conexión a internet, creando por ejemplo una conexión ethernet.

Bibliografía.

[1] Dª Raquel Gutiérrez Rivas, "*People Monitoring in Intelligent Spaces for the management of Health and Comfort*", .Capítulo 3.

[2] OpenCourseWare impartido por la universidad de Valencia. "*Bioseñales*". Tema 2. [Online]. Disponible:

http://ocw.uv.es/ingenieria-y-arquitectura/1-5/ib_material/IB_T2_OCW.pdf.

[Último Acceso: Septiembre-2014]

[3] My EKG. La Web del Electrocardiograma. "*Intervalos y segmentos del electrocardiograma*". [Online]. Disponible:

http://www.my-ekg.com/generalidades-ekg/intervalos-segmentos-ekg.html

[Último Acceso: Octubre-2014]

[4] OpenCourseWare impartido por la universidad de Valencia. "*Series Temporales*". Tema 6. [Online]. Disponible:

http://ocw.uv.es/ingenieria-y-arquitectura/1-5/ib_material/IB_T6_OCW.pdf.
[Último Acceso: Septiembre-2014]

[5] ING. Humberto González. *"Señales Biomédicas. Parte 2".* [Online]. Disponible:

http://www.angelfire.com/un/biomedicafime/CLASE_5.pdf.

[Último Acceso: Septiembre-2014]

[6] Cajavilca-C, Varon-J, Herrero-S. "*Historia de la medicina: Willem Einthoven y la Aplicación Clínica del Electrocardiograma".* [Online]. Disponible:

http://hgculiacan.com/revistahgc/archivos/Rev6%20Historia%20de%20la%20Medicina.pdf. [Último Acceso: Octubre-2014]

[7] Sergio Hinojosa. "*Detector de QRS basado en el algoritmo de Pan y Tompkins".* [Online]. Disponible:

file:///C:/Users/Javier/Downloads/qrsdetector%20(1).pdf.

[Último Acceso: Noviembre-2014]

[8] Xilinx. "*Virtex-5 FPGA User Guide".* Ug190(v5.4) Marzo 16, 2012.

[9] Xilinx. "*Virtex-5 FPGA XtremeDSP Design Considerations. User Guide"* UG193(v3.5) Enero 26, 2012.

[10] OpenCourseWare impartido por la universidad de Valencia. "*Arquitectura de los procesadores digitales de señal*". Tema 3. [Online]. Disponible:

http://ocw.uv.es/ingenieria-y-arquitectura/sistemas-electronicos-para-el-tratamiento-de-la-informacion/seti_materiales/seti3_ocw.pdf

[Último Acceso: Noviembre-2014]

[11] Xilinx. "*DSP: Designing for Optimal Results. High Performance DSP Using Virtex-4 FPGAs*" Edition 1.0 Marzo 2005.

[12] Apuntes de la asignatura "*Diseño electrónico*" Master en sistemas electrónicos avanzados. Sistemas inteligentes, Politécnica de Alcalá. .Autor Álvaro Hernández Alonso.

[13] S. S. Barold, "*Willem Einthoven and the Birth of Clinical Electrocardiography a Hundred Years Ago*," Card. Electrophysiol. Rev., vol. 7, no. 1, pp. 99–104, 2003

[14] J. Pan and W. J. Tompkins, "A *real-time QRS detection algorithm.,*" *IEEE Trans. Biomed. Eng.*, vol. 32, no. 3, pp. 230–6, Mar. 1985.